

Self-energy Feynman Diagrams with Four Loops and 11 Internal Lines

Elise de Doncker[1(✉)] and Fukuko Yuasa[2]

[1] Western Michigan University, Kalamazoo, MI 49008, USA
elise.dedoncker@wmich.edu
[2] High Energy Accelerator Research Organization (KEK),
1-1 OHO, Tsukuba, Ibaraki 305-0801, Japan
fukuko.yuasa@kek.jp
http://www.cs.wmich.edu/elise

Abstract. We consider 4-loop Feynman diagrams with 11 internal lines. The associated 10-dimensional loop integrals are calculated for four diagrams with massive internal lines, and we further handle the massless case of the diagram referenced in the literature as $M61$. The computations are performed with double exponential (DE), Quasi-Monte Carlo (lattice and embedded lattice rules) and adaptive integration algorithms, which do not require any user input regarding the integrand behavior. The lattice rule methods are combined with a transformation to help alleviate boundary singularities. The embedded lattice rules are implemented in CUDA C and their execution is accelerated using an NVIDIA Quadro GV100 GPU, whereas DE is parallelized over MPI and executed on an AMD cluster. Adaptive integration is performed with the ParInt multivariate integration package, which is also layered over MPI. For the massless $M61$ diagram we use a dimensional regularization approach and extrapolation. The results will be compared with respect to accuracy and efficiency, and verified with pySecDec.

Keywords: Feynman loop diagrams · Lattice rules · Adaptive integration · High performance computing · Dimensional regularization

1 Introduction

The Standard Model of particle physics developed in the last century is well established and has explained a wide range of physics phenomena. However, there remain several problems for which the model does not provide an explanation. For example, a theoretical prediction of the muon magnetic anomaly in the Standard Model provides an excellent result [2]. However, a recent highly precise measurement at Fermi National Accelerator Laboratory (FNAL) revealed that the discrepancy between experiment and theory has increased [1]. Not only by the anomaly but also by other unexplained problems, the limitation of the Standard Model has been recognized.

Therefore, precise tests of the Standard Model are inevitable and it is expected that further limitations will emerge from future experiments. In response, precise theoretical predictions are absolutely necessary. This means that a precise calculation of higher-order corrections in perturbative theory is required. Over the past two decades, significant advances have been made in analytical, semi-analytical and numerical methods

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12953, pp. 160–175, 2021.
https://doi.org/10.1007/978-3-030-86976-2_11

in this research field. One of challenges is a precise evaluation of multi-loop Feynman integrals, which appear as a key ingredient in the calculations.

We have been developing fully numerical methods for the evaluation of multi-loop integrals with sufficient accuracy. Generally speaking, numerical approaches are computationally intensive. However, our methods succeed in performing the computation in a realistic time with the help of modern computer architectures. In this contribution, we will briefly describe our techniques and present recent developments taking 4-loop Feynman diagrams with 11 lines as an example.

Subsequently in this paper, we review background information on Feynman loop diagrams/integrals and on the numerical methods used, in Sects. 2 and 3, respectively. After an outline of automatic integration in Sect. 3.1, non-adaptive methods are covered in Sect. 3.2, including the double exponential (DE) method, and Quasi-Monte Carlo (QMC) approaches based on lattice rules, embedded lattice rules and stochastic rule families. The adaptive algorithm from the ParInt package is discussed briefly in Sect. 3.3. Sections 3.4 and 3.5 further specify the transformations used and the extrapolation, respectively. Results for the diagrams of Fig. 1 are reported in Sect. 4, including those for the massive diagrams in Sect. 4.1, and the massless diagram in Sect. 4.2. Appendix A.1–A.4 lists the main parts of the integrand functions.

2 Feynman Diagrams and Loop Integrals

A general form of the scalar Feynman integral in Feynman parameter (Euclidean) space is given by $\mathcal{F} = (4\pi)^{-\nu L/2}\mathcal{I}$ with

$$\mathcal{I} = \Gamma(N - \frac{\nu L}{2})\,(-1)^N \int_{\mathcal{C}_N} \prod_{r=1}^{N} dx_r\, \delta(1 - \sum x_r) \frac{C^{N-\nu(L+1)/2}}{(D - i\varrho C)^{N-\nu L/2}} \tag{1}$$

$$= \Gamma(N - \frac{\nu L}{2})\,(-1)^N \int_{\mathcal{S}_{N-1}} \prod_{r=1}^{N-1} dx_r\, \frac{C^{N-\nu(L+1)/2}}{(D - i\varrho C)^{N-\nu L/2}} \tag{2}$$

and where L is the number of loops, N is the number of internal lines, and ρ is an infinitesimal parameter preventing the denominator from vanishing. In the integrand function, C and D are polynomials depending on physical parameters and the topology of the diagram. For the instances covered here, D does not vanish in the interior of the domain, thus we have $\varrho = 0$. The space-time dimension ν is expressed in terms of a parameter ε, which approaches zero for regularization, $\nu = 4 - 2\varepsilon$. By eliminating the δ-function in (1), the N-dimensional unit cube integration domain $\mathcal{C}_N$ transforms into the $d = (N-1)$-dimensional unit simplex $\mathcal{S}_d = \{\mathbf{x} \in \mathcal{C}_d \mid \sum_{j=1}^{d} x_j \le 1\}$ in (2).

We further incorporate the normalization factor $n(\varepsilon)^4$ where

$$n(\varepsilon) = \frac{\Gamma(2 - 2\varepsilon)}{\Gamma(1 + \varepsilon)\,\Gamma(1 - \varepsilon)^2} \tag{3}$$

The C and D functions are given in Appendix A.1–A.4.

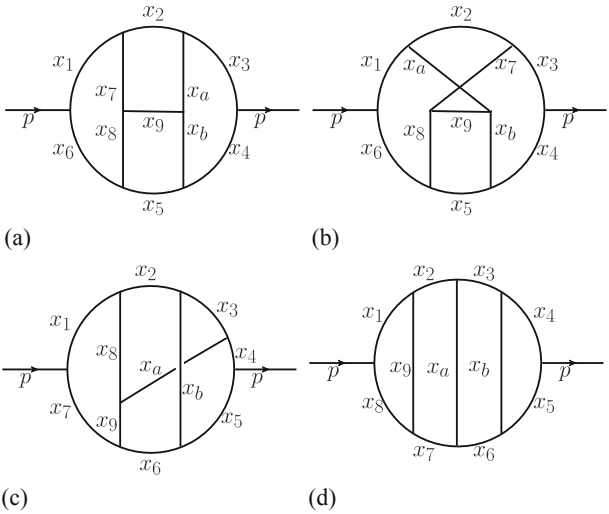

Fig. 1. 4-loop $N = 11$ diagrams (a) $M61$ [3,8,26,38]; (b) $M62$ [3,38]; (c) $M63$ [3,38]; (d) *BH* [5]

2.1 Loop Diagram Expansions

We will treat the $(L = 4)$-loop self-energy diagrams of Fig. 1 with $N = 11$ internal lines, thus the integrand behaves as $C^{1+5\varepsilon} \,/\, D^{3+4\varepsilon}$. For massive lines with all masses $= 1$, and $s = p^2 = 1$, we give results in this paper for all diagrams of Fig. 1, whereas for $M61$ in Fig. 1(a) we also handle the diagram with massless internal lines. The massless case of $M61$ (Fig. 1(a)) was considered in [8,26]. We covered the massless diagram of Fig. 1(d) in [20].

The integrals for the massless diagrams of $M61, M62$ and $M63$ satisfy an expansion in ε of the form

$$\mathcal{I} \sim \mathcal{I}(\varepsilon) = \gamma(\varepsilon) \sum_{k \geq \kappa} C_k \varepsilon^k \tag{4}$$

where γ is a factor that depends on ε, and $\kappa = -1$. Specifically, analytic results for these diagrams have been presented in the literature as an expansion in powers of ε with coefficients expressed in terms of Riemann zeta functions as follows.

***M61*, Fig. 1(a).** We compute results for the massless case according to [3,34,38]

$$\begin{aligned}\mathcal{I} \sim n(\varepsilon)^4 \Gamma(3+4\varepsilon)\,(&-\frac{10\,\zeta_5}{\varepsilon} + (50\,\zeta_5 - 10\,\zeta_3^2 - 25\zeta_6) \\ &+ (90\,\zeta_5 + 50\,\zeta_3^2 + 125\zeta_6 - 30\,\zeta_3\,\zeta_4 + 9.5\,\zeta_7)\,\varepsilon + \mathcal{O}(\varepsilon^2))\end{aligned} \tag{5}$$

***M62*, Fig. 1(b).** Known results are from [3,38], corresponding to

$$\mathcal{I} \sim n(\varepsilon)^4 \Gamma(3+4\varepsilon)\,(-\frac{10\,\zeta_5}{\varepsilon} + (130\,\zeta_5 - 10\,\zeta_3^2 - 25\zeta_6 - 70\zeta_7) + \mathcal{O}(\varepsilon)) \tag{6}$$

***M*63, Fig. 1(c).** The expansion for the massless case is [3,38]

$$\mathcal{I} \sim n(\varepsilon)^4 \Gamma(3+4\varepsilon)\,(-\frac{5\,\zeta_5}{\varepsilon} + (45\,\zeta_5 - 41\,\zeta_3^2 - 12.5\zeta_6 + 80.5\zeta_7) + \mathcal{O}(\varepsilon)) \tag{7}$$

***BH*, Fig. 1(d).** The diagram (from [5]) is finite, with $\kappa = 0$ in (4). In [20] we gave numerical results for the constant C_0 in the expansion

$$\mathcal{I} \sim \Gamma(3+4\varepsilon)\,(C_0 + C_1\varepsilon + \mathcal{O}(\varepsilon^2)) \tag{8}$$

using lattice rules and adaptive methods for the massless case, where the analytic value is $C_0 = 35\,\zeta_7 \approx 35.2922$.

3 Numerical Methods

This section provides background information (as in [20]) on the termination criterion in automatic integration, non-adaptive and adaptive strategies, the transformations used, and extrapolation by the ϵ-algorithm [35,42].

3.1 Automatic Integration

Automatic integration methods act like a black-box, where the input includes the integrand specification $f(\mathbf{x})$, dimension (number of function variables) d and domain $\mathcal{D}$, with the objective to compute an approximation $\mathcal{Q}$ to an integral of the form

$$\mathcal{I} = \int_{\mathcal{D}} f(\mathbf{x})\,d\mathbf{x} \tag{9}$$

An (absolute) error estimate E_a is usually provided as well. To realize a termination criterion, requested absolute and relative accuracies (tolerances) t_a and t_r, respectively, and/or a maximum number of interactions or function evaluations may further be specified by the user. The algorithm then performs a prescribed number of iterations, or attempts to satisfy an accuracy requirement such as that of the integration packages Quadpack [32] and ParInt [15,16,19],

$$|\mathcal{Q} - \mathcal{I}| \le E_a \le t = \max\{t_a, t_r|\mathcal{I}|\}. \tag{10}$$

3.2 Non-adaptive Integration

Double-Exponential Method. The double-exponential method (DE) [39,40] transforms the one-dimensional integral

$$\int_0^1 f(x)\,dx = \int_{-\infty}^{\infty} f\,(\phi(t))\,\phi'(t)\,dt, \quad \text{using}$$
$$x = \phi\,(t) = \frac{1}{2}(\tanh\,(\frac{\pi}{2}\sinh\,(\mathrm{t})) + 1), \quad \phi'(\mathrm{t}) = \frac{\pi\cosh\,(\mathrm{t})}{4\cosh^2(\frac{\pi}{2}\sinh\,(\mathrm{t}))} \tag{11}$$

Due to the rapid decrease of the integrand (as $|t|$ increases) after the transformation, the infinite range can be truncated and a trapezoidal rule approximation may be suitable, provided the integrand is smooth inside the domain. This yields

$$I_h^{N_{eval}} = \sum_{k=-N_-}^{N_+} f(\phi(kh))\,\phi'(kh) \tag{12}$$

with mesh size h and $N_{eval} = N_- + N_+ + 1$ function evaluations. For multivariate integration, a product-wise application of formula (11) is used.

Lattice Rules. Rank-1 lattice rules provide an integral approximation over the unit cube of the form

$$\mathcal{Q}f = \mathcal{Q}(\mathbf{z}, n)f = \frac{1}{n}\sum_{j=0}^{n-1} f(\{\frac{j}{n}\mathbf{z}\}) \tag{13}$$

Here $\mathbf{z}$ is a generator vector with components $z \in \mathcal{Z}_n = \{1 \le z < n \mid \gcd(z, n) = 1\}$, where gcd denotes the greatest common divisor, and $\{\mathbf{x}\}$ is a vector obtained by taking the fractional part of each component of $\mathbf{x}$ (see [37]). Classically n was taken prime [23,24] (so that $\mathcal{Z}_n = \{1, 2, \ldots, n-1\}$); extensions to non-prime n are given in [29,31]. We precomputed the generators $\mathbf{z}$ for various n in [12,13,17,18], using the component by component (CBC) algorithm [30,31] either directly or via Lattice Builder [25], including the rules with 400,000,009 and 5^{13} points used for the results in Sect. 4 of this paper.

A sequence embedding of lattice rules is defined in [37], based on a rank-1 rule Q_0 and where each rule of the sequence applies a scaled copy of Q_0 to the subregions obtained by subdividing $\mathcal{C}_d$ into m equal parts in each of r coordinate directions. Denoting

$$\mathcal{Q}_r f = \mathcal{Q}_r^{(m)}(\mathbf{z}, n)f = \frac{1}{m^r n}\sum_{k_r=0}^{m-1} \cdots \sum_{k_1=0}^{m-1}\sum_{j=0}^{n-1} f\left(\{\frac{j}{n}\mathbf{z} + \frac{1}{m}(k_1, \ldots, k_r, 0, \ldots, 0)\}\right) \tag{14}$$

for n and m relatively prime, the points of Q_r are embedded in Q_{r+1} for $0 \le r < d$. Q_r has $m^r n$ points and is of rank r for $1 \le r \le d$, and Q_d is composed of m^d scaled copies of Q_0. An error estimate for Q_d is calculated based on a sequence of rules $Q^{(i)}$, $1 \le i \le d$, of order $m^{d-1}n$ and embedded in Q_d [37]. The results for the 10-dimensional integrals ($d = 10$) in Sect. 4 of this paper are obtained with $m = 2$.

A stochastic family of rules can be considered as proposed by Cranley and Patterson [10], by adding a random vector $\boldsymbol{\Delta}$ to the integration points,

$$\mathcal{Q}(\mathbf{z}, n, \boldsymbol{\Delta})f = \frac{1}{n}\sum_{j=0}^{n-1} f(\{\frac{j}{n}\mathbf{z} + \boldsymbol{\Delta}\}) \tag{15}$$

Averaging over a number of randomized (shifted) rule sums of the form (15) where $\boldsymbol{\Delta}$ has a multivariate uniform random distribution on $\mathcal{C}_d$ yields an integral approximation that has the integral $\mathcal{I}$ as its expected value [37]. An error estimate is given in terms of the standard deviation. This technique is applied by pySecDec [6].

3.3 Adaptive Multivariate Integration — ParInt

Adaptive automatic integration software includes HALF [11], Adapt [21], Dcuhre [4], Cuba [22], adaptive programs in Quadpack [14,19,32] and ParInt [15,16,19]. The basic global adaptive strategy follows the algorithm of Fig. 2, cf., [27,33]. Our accuracy requirement for the global adaptive algorithm is given by (10), and the termination criterion tested in the course of the computations is of the form $E_a \le t = \max\{t_a, t_r|\mathcal{Q}|\}$, where $\mathcal{Q}$ is the current overall integral approximation.

```
Evaluate given region and initialize results
Initialize priority queue to empty
while (evaluation limit is not reached
        and estimated error is too large)
    Retrieve region from priority queue
    Split region
    Evaluate subregions and update results
    Insert subregions into priority queue
```

Fig. 2. Adaptive region/task partitioning strategy

ParInt is a parallel automatic integration method implemented in C and layered over MPI (Message Passing Interface [28]). The user-specified domain is distributed over the participating MPI processes (workers), each of which successively selects and deletes the largest error subregion from its local priority queue (heap), subdivides this region, integrates over the parts, and inserts the children regions into the heap. The updates to the integral and error estimate are accumulated, and communicated with the controller process. A load balancing procedure further allows sending work from busy to idle or near-idle workers. Based on the distributed task partition strategy where each task processes one or more regions, the region subdivison algorithm tends to refine the domain around integration hotspots such as peaks, ridges and singularities.

3.4 Transformations

Since we rely on the loop integral representation (2) over a simplex domain, a transformation is needed from the simplex to the unit cube. Unless otherwise stated we transform the integral over $\mathcal{S}_d$ of (2) to $\mathcal{C}_d$ by the coordinate transformation

$$x_j = (1 - \sum_{k=1}^{j-1} x_k)\, y_j, \quad 1 \le j \le d \tag{16}$$

so that $\mathbf{y} \in \mathcal{C}_d$ and the Jacobian is $\prod_{j=1}^{d-1}(1 - \sum_{k=1}^{j-1} x_k)$.

Furthermore, while lattice rules are applied with a periodizing transformation of the integrand, we select a transformation that also suppresses singularities on the boundaries of the domain. We make use of Sidi's $\sin^m$-transformations [36] for $m = 6$,

$$\Psi_6(t) = t - (45\sin(2\pi t) - 9\sin(4\pi t) + \sin(6\pi t))/(60\pi), \quad \Psi_6'(t) = \frac{16}{5}\sin^6(\pi t) \tag{17}$$

3.5 Extrapolation

With the goal of obtaining the limit numerically, we use the ϵ-algorithm [35,42] for a nonlinear extrapolation, and an implementation based on the code of *qext()* from the QuadPack package [32]. The computation produces a triangular table, with a sequence of integral approximations as the entry sequence in the first column. The *qext()* procedure computes a new lower diagonal of the table when a new entry is available, together with a measure of the distance of each element from preceding neighboring elements. In the construction of the triangular table, a table element *"new"* relative to preceding e_0, e_1, e_2, e_3 is obtained as

$$\begin{matrix} & e_0 & \\ e_3 & e_1 & new \end{matrix} = e_1 + 1/(1/(e_1 - e_3) + 1/(e_2 - e_1) - 1/(e_1 - e_0)) \qquad (18)$$

$$\begin{matrix} & e_2 & \end{matrix}$$

and the distance for *new* is set to

$$distance = |e_2 - e_1| + |e_1 - e_0| + |e_2 - new| \qquad (19)$$

Then *qext()* returns the new lower diagonal element with the smallest value of the distance estimate as the extrapolation result.

4 Numerical Results

4.1 Massive Diagrams

The GPU results were obtained on an x86_64 machine at KEK, with Intel(R) Xeon(R) Gold (dual) 6230 CPU@2.10 GHz, 20 cores (40 HT−hyper-threaded), under GNU/Linux, with NVIDIA Quadro GV100 GPU and using CUDA C. The Quadro GV100 is a Volta architecture with compute capability 7.0, and with 5,120 cores and 640 tensor cores, 32 GB HBM2 (High Bandwidth Memory second generation) global memory, (boost) clock rate of 1,627 MHz, and memory clock rate 848 MHz. It performs at 33.32 TFLOPS for FP16 (half), 16.66 TFLOPS for FP32 (float), and 8.330 TFLOPS for FP64 (double) (from [41] and also using a device query program). The results in Tables 1, 2, 3, and 4 are computed in double precision.

The methods include *MPI + DE* and *OpenMP + Sidi 6*, which are run on AMD cluster nodes with a Ryzen Threadripper 3990X 64-Core (128 HT) CPU. According to [9], this CPU has a base clock speed of 2.9 GHz, observed boost clock speed of 3.4 GHz, and maximum boost clock speed of 4.3 GHz with lightly threaded workloads. *MPI + DE* employs 19 processes on one node, where DE evaluates 19 points in each coordinate direction. *OpenMP + Sidi 6* is multithreaded with 128 threads, for a lattice rule with $\approx 400M$ points and Sidi's $\sin^6$ (Ψ_6) transformation. Ψ_6 is also used for the *Sidi 6 Quadro GPU* results with $400M$- and 5^{13}-point lattice rules, and for the *Sidi 6* $m = 2$ *Quadro GPU* embedded (2^{10}-copy) versions of the lattice rules with two copies in each coordinate direction. pySecDec 1.4.4 [6,7] comparison results are obtained on an AMD node (with 128 threads) for a Korobov-3 (shifted) rule, without use of a GPU.

Tables 1, 2, 3, and 4 display the resulting integral approximations, number of integrand evaluations, time in seconds, and an absolute error estimate (E_a) for the methods

Table 1. Results of DE, lattice and embedded lattice rules for $M61$ massive case (Fig. 1(a))

Method	Evaluations	Result	E_a	Time[s]
MPI + DE	$19^{10} = 6.131T$	−2.862726657918146 e-02		35,102
OpenMP + Sidi6	$400M$	−2.8596810730 e-02		2.70
Sidi6 Quadro GPU	$400M$	−2.859681073072325 e-02		0.23
Sidi 6 Quadro GPU	5^{13}	−2.863834054956898 e-02		0.66
Sidi 6 $m = 2$ Quadro	$400M \times 2^{10}$	−2.862717083713596 e-02	4.72 e-08	209.2
Sidi 6 $m = 2$ Quadro	$5^{13} \times 2^{10}$	−2.862717128087744 e-02	1.65 e-09	666.5
pySecDec Korobov-3	$337,940,896$	−2.862716229785056 e-02	5.59 e-09	117,046

Table 2. Results of DE, lattice and embedded lattice rules for $M62$ massive case (Fig. 1(b))

Method	Evaluations	Result	E_a	Time[s]
MPI + DE	$19^{10} = 6.131T$	−2.549545550207116 e-02		45,042
OpenMP + Sidi 6	$400M$	−2.5466771436 e-02		2.71
Sidi 6 Quadro GPU	$400M$	−2.546677143565166 e-02		0.23
Sidi 6 Quadro GPU	5^{13}	−2.551194982708940 e-02		0.68
Sidi 6 $m = 2$ Quadro	$400M \times 2^{10}$	−2.549545504404488 e-02	8.39 e-08	219.5
Sidi 6 $m = 2$ Quadro	$5^{13} \times 2^{10}$	−2.549545504820467 e-02	6.94 e-10	694.4
pySecDec Korobov-3	$337,940,896$	−2.549544878067482 e-02	4.57 e-09	142,359

Table 3. Results of DE, lattice and embedded lattice rules for $M63$ massive case (Fig. 1(c))

Method	Evaluations	Result	E_a	Time[s]
MPI + DE	$19^{10} = 6.131T$	−2.765508186953987 e-02		46,775
OpenMP + Sidi6	$400M$	−2.7643999940 e-02		2.81
Sidi6 Quadro GPU	$400M$	−2.764399993954736 e-02		0.24
Sidi 6 Quadro GPU	5^{13}	−2.766877323614956 e-02		0.69
Sidi 6 $m = 2$ Quadro	$400M \times 2^{10}$	−2.765491227212118 e-02	1.38 e-07	223.1
Sidi 6 $m = 2$ Quadro	$5^{13} \times 2^{10}$	−2.765491019803990 e-02	1.08 e-09	706.9
pySecDec Korobov-3	$337,940,896$	−2.765491546054626 e-02	5.61 e-09	119,274

Table 4. Results of DE, lattice and embedded lattice rules for *BH* massive case (Fig. 1(d))

Method	Evaluations	Result	E_a	Time[s]
MPI + DE	$19^{10} = 6.131T$	−9.341399988896524 e-02		29,845
OpenMP + Sidi 6	$400M$	−9.3338131744 e-02		2.67
Sidi 6 Quadro GPU	$400M$	−9.333813174440837 e-02		0.23
Sidi 6 Quadro GPU	5^{13}	−9.341291671067280 e-02		0.62
Sidi 6 $m = 2$ Quadro	$400M \times 2^{10}$	−9.340329476470503 e-02	4.34 e-07	201.1
Sidi 6 $m = 2$ Quadro	$5^{13} \times 2^{10}$	−9.340331509556005 e-02	5.20 e-08	638.9
pySecDec Korobov-3	$337,940,896$	−9.340331518736177 e-02	1.08 e-08	5,092

that supply a value for E_a. Based on this comparison and the estimated error, there is good agreement between the embedded lattice rules (*Sidi 6* $m = 2$ *Quadro*), whereas the esimated error of the embedded 5^{13}-point rule is smaller. There is also good agreement between pySecDec and the embedded rules, in particular the embedded 5^{13}-point rule, but the latter is much more efficient with respect to execution times. *MPI + DE* is less accurate than pySecDec but faster, except for the *BH* diagram. We observe that its result for $M62$ agrees with the embedded lattice rules to 4.5×10^{-10}. As expected, the single lattice rule methods (*OpenMP + Sidi 6* and *Sidi 6 Quadro GPU*) are very fast but provide low accuracy in comparison.

4.2 Results for Massless $M61$

For the diagrams with massless internal lines, the space-time dimension is $\nu = 4 - 2\varepsilon$, which yields $\mathcal{I}(\varepsilon)$ with $C^{1+5\varepsilon}/D^{3+4\varepsilon}$ under the integral sign in (2). The massless case of $M61$ was covered in [8,26], where [8] includes numerical results obtained via sector decomposition.

In this section we will compute the coefficient of $1/\varepsilon$ in the expansion (5) for $M61$ by approximating the limit of $\varepsilon\mathcal{Q}(\varepsilon)$ as $\varepsilon \to 0$. This gives a basis for dimensional regularization where $\nu \to 4$ as $\varepsilon \to 0$. The convergence of the sequence $\varepsilon\mathcal{Q}(\varepsilon)$ is accelerated by the ϵ-algorithm [35,42] (cf., Sect. 3.5).

Tables 5 and 6 list the sequences $\varepsilon_\ell \mathcal{Q}^{(2)}(\varepsilon_\ell)$ for $\varepsilon_\ell = -1/b^\ell$ with $b = 1.2$ in Table 5 and $b = 1.15$ in Table 6. The integral approximation $\mathcal{Q}^{(2)}$ is an embedded $400M$-point lattice rule $\mathcal{Q}$ with $m = 2$, implemented using Sidi 6 on Quadro GPU. Also listed are the corresponding absolute error estimate E_a, execution time in seconds and the extrapolated sequence. The latter consists of elements selected from the lower diagonal of the triangular extrapolation table, computed according to (18).

As the estimated absolute error of the entry sequence increases to the order of 10^{-2}, this will limit the accuracy of the extrapolated result. The first extrapolation occurs at the third iteration (with three elements in the entry sequence), i.e., at $\ell = 7$ in Table 5 and at $\ell = 9$ in Table 6. The exact integral is $-10\,\zeta_5 \approx -10.369277551433699$. An accurate extrapolated result is reached at the 7th iteration (with five extrapolations), which is at $\ell = 11$ in Table 5 and at $\ell = 13$ in Table 6.

We further obtained results for this case with adaptive integration in ParInt, on the *thor* cluster in the Department of Computer Science at WMU. The MPI runs were performed on four Intel Xeon E5-2670, 2.6 GHz dual CPU nodes with 16 processes per node. The integral approximation, absolute error estimate, and time in seconds for the adaptive integration are given in Table 7, with their extrapolated values for a sequence $\varepsilon_\ell \mathcal{Q}(\varepsilon_\ell)$ with $\varepsilon_\ell = -1/b^\ell$, $b = 1.2$ and $\ell = 3, 4, \ldots$. The maximum number of integrand evaluations was set at $500B$.

The first sequence of extrapolation results (column 5) in Table 7 applies to the entry sequence starting at $\ell = 3$. Although the first three steps are erratic, convergence sets in after that. In fact, at $\ell = 7$, two results were available on the lower diagonal, including -17.85705981951788 besides the $169.7\ldots$ value listed. The latter was selected as it appeared with a slightly smaller *distance* estimate as given by (19).

The second sequence of extrapolation results (column 6) shows the behavior when the entry sequence is started at $\ell = 6$. Note that the second extrapolation, as it is started

Table 5. Results of embedded lattice rule and extrapolation for $M61$ massless case (Fig. 1(a)), $400M$-point rule, $m = 2$, sequence with $\varepsilon_\ell = -1/b^\ell, b = 1.2, \ell = 5, 6, \ldots$

ℓ	$\varepsilon_\ell \mathcal{Q}^{(2)}(\varepsilon_\ell)$	E_a	Time[s]	Extrapolated
5	−5.8670814834389446	4.045e-04	236.9	
6	−6.6010810209390192	9.334e-04	247.6	
7	−7.3290025451281524	1.736e-03	247.8	−94.50711787172600
8	−8.0025461298639637	2.795e-03	248.2	−16.34528520934288
9	−8.5955073617360025	4.044e-03	248.1	−12.95878306674803
10	−9.0962473167603335	5.377e-03	248.2	−11.81514810764655
11	−9.5035175173401605	6.665e-03	248.3	−10.33468561083054
12	−9.8237642007586796	7.636e-03	248.2	−10.30358698496245
13	−10.076829148927793	1.087e-02	248.3	−10.33470169032785

Table 6. Results of embedded lattice rule and extrapolation for $M61$ massless case (Fig. 1(a)), $400M$-point rule, $m = 2$, sequence with $\varepsilon_\ell = -1/b^\ell, b = 1.15, \ell = 7, 8, \ldots$

ℓ	$\varepsilon_\ell \mathcal{Q}^{(2)}(\varepsilon_\ell)$	E_a	Time[s]	Extrapolated
7	−6.1319925400428952	5.674e-04	236.2	
8	−6.6992591321666515	1.024e-03	247.6	
9	−7.2574395641246294	1.642e-03	248.2	−41.54754549845097
10	−7.7853886548937465	2.415e-03	248.4	−17.00529857230883
11	−8.2697056028975808	3.317e-03	248.7	−13.64562553847281
12	−8.7025765235833710	4.306e-03	249.2	−12.34478650059442
13	−9.0804943525783131	5.331e-03	249.5	−10.29791987579131
14	−9.4032831930914433	6.333e-03	249.4	−10.31245440302037
15	−9.6733449900103512	7.227e-03	249.2	−10.23243241946978
16	−9.8958550922968769	7.739e-03	249.3	−10.30299005253333
17	−10.084346379310391	1.145e-02	249.7	−10.35978087121498

at a later element of the entry sequence (column 1), generates a triangular ϵ-algorithm table that is a subtriangle of that started at an earlier entry element (as for the extrapolation sequence in column 5). However, the extrapolated value is selected at each iteration as the element with lowest distance estimate along the lower diagonal of the triangular table. This explains why some of the extrapolated values in columns 5 and 6 agree, while other ones are different.

While the extrapolation results are good, the adaptive integration is outperformed with respect to execution time by the lattice rules, which are indeed very efficient for this type of problem.

Table 7. Results of adaptive integration (ParInt) and extrapolation for $M61$ massless case (Fig. 1(a)), $500B$ integrand evaluations, sequence with $\varepsilon_\ell = -1/b^\ell, b = 1.2, \ell = 3, 4, \ldots$

ℓ	$\varepsilon_\ell \mathcal{Q}^{(2)}(\varepsilon_\ell)$	E_a	Time[s]	Extr. start $\ell = 3$	Extr. start $\ell = 6$
3	−4.91021341679181	0.00179	3292.7		
4	−5.20347093132543	0.00231	3310.5		
5	−5.84191584324681	0.00461	2818.2	−4.661073469965100	
6	−6.57615789432883	0.00275	2842.3	−0.948522306243995	
7	−7.31346988965485	0.00230	3371.6	169.7675893230898	
8	−7.98259667712166	0.00327	3385.2	−14.54898654030341	−14.54898654030341
9	−8.57201855470499	0.00501	3394.9	−12.93082342985936	−12.93082342985936
10	−9.06743566419856	0.00697	3398.1	−11.67834697610792	−11.67834697610792
11	−9.47072913855170	0.00826	3398.8	−11.26526383021044	−10.80236320496894
12	−9.78754291117960	0.01620	3398.0	−10.35212751126703	−9.900013844098643
13	−10.0284149761477	0.02240	3393.0	−10.22374747556268	−10.50801809515002
14	−10.2029760970888	0.03150	2876.7	−10.36314903844882	−10.36314903844882

5 Conclusions

We reported results incurred for diagrams with four loops and 11 internal lines, $M61$, $M62$, $M63$, and a diagram we call *BH* with reference to Binoth and Heinrich [5]. We employed automatic integration methods using QMC/lattice rules and adaptive integration, where the user is not required to specify characteristics of the integrand function − except for the function definition itself. It is clear from the function definitions in Appendix A.1–A.4 that it may be difficult to guage the actual integrand behavior.

The adaptive method is less efficient than the QMC embedded 2^d lattice copy rule in the dimensional regularization approach for the massless $M61$ problem. However, adaptive integration may be more versatile in general. The QMC embedded 2^d lattice copy rules compare favorably to pySecDec with respect to accuracy (which we used as a yardstick), but are much more efficient than the multithreaded pySecDec for these types of integrals.

To conclude, we observed that our fully numerical methods deliver good performance for computations of Feynman diagrams with four loops and 11 internal lines, with respect to accuracy and elapsed time. For further improvements, we will study suitable variable transformations in future work.

Acknowledgments. We acknowledge the support of the Grant-in-Aid for Scientific Research (JP17K05428 and JP20K03941) from JSPS KAKENHI, as well as the National Science Foundation Award Number 1126438 that funded the cluster used for the computations with ParInt in this paper. We also thank our colleagues, Dr. T. Ishikawa, Dr. H. Daisaka, Dr. N. Nakasato and Dr. J. Kapenga, for creating and maintaining computing environments supporting our Feynman integration work.

A C and D functions

The C and D functions for the diagrams of Fig. 1 are given in C code form below. We denote

$$x_{i_1 \ldots i_k} = \sum_{j=i_1}^{i_k} x_j \tag{20}$$

We use $x_a = x_{10}$ and $x_b = x_{11}$. Obvious notations also include $xksq = x_k^2$.

A.1 $M61$, Fig. 1(a).

```
  C = (x16*x5*x79 + x56*(x79*x8 + x7*x9) +
     x2*(x5*x678 + x67*x89 + x8*x9) +
     x1*(x2*x58 + x7*x89 + x23*x9 + x4*x9 + x8*x9) +
     x34*(x7*x89 + x6*x9 + x8*x9))*xab +
     (x16*x234578 + x2345*x78)*xa*xb +
     x34*(x1*x2*x5 + x5*(x6*x79a + x7*x89a) + x56*x8*x9a +
     x1*(x5*x79a + x8*x9a) + x16*x27*x89b +
     x2*(x7*x8 + x5*x678 + x78*x9b));

  D1 = x34*(x1*x2sq*x589b + 2*x1*x2*(x589b*x7 + x8*x9) +
     x1sq*(x2*x589b + x5*x79a + x7*x89b + x8*x9a) +
     x2sq*(x5*x678 + x7*x8 + x6*x89b + x78*x9b)) +
     2*(x1*(x2*x3*x5 + x3*x5*x7) + x2*x3*(x5*x678 + x6*x8))*xa +
     x3sq*(x6*(x7*x89b + x8*x9a) +
     x1*(x2*x589b + x5*x79a + x7*x89b + x8*x9a) +
     x2*(x5*x678 + x7*x8 + x6*x89b + x78*x9b) +
     x5*(x6*x79 + x7*x89 + x8*x9 + x678*xa)) +
     (x1*x2sq*x589 + x1sq*(x27*x58 + x234578*x9) +
     2*x2*x3*(x7*x8 + x678*x9) + x3sq*(x7*x8 + x1*x9 + x678*x9) +
     2*x1*x2*(x589*x7 + x3*x89 + x8*x9) +
     x2sq*(x5*x678 + x67*x89 + x8*x9) +
     2*x1*x3*(x7*x8 + x7*x9 + x8*x9))*xab +
     (x1*x2sq + x1sq*x234578 + x1*x3sq + x23sq*x678 +
     2*x1*(x2*x378 + x3*x78))*xa*xb;

  D = D1 + C*x456789ab; // Massive diagram
//D = D1 - C*x123;      // Massless diagram
```

A.2 $M62$, Fig. 1(b).

```
  C = x1*(x3*(x5*x79a + x7*x89b + x8*x9a) +
     x2*(x34*x589b + x58*x7 + x578*x9b) + x4*(x58*x79a + x7*x9b) +
     (x58*x7 + x34578*x9)*xab + x34578*xa*xb) +
     x2*(x58*x6*x7 + x5*x7*x89a + x578*x6*x9b + x5*x8*x9b +
     (x7*x8 + x578*x9)*xab + x34*(x5*x689a + x6*x89b + x89*xab) +
     x34578*xa*xb) + x34*(x5*(x6*x79a + x7*x89a + x8*x9a) +
     x6*(x7*x89b + x8*x9a) + (x7*x8 + x678*x9)*xab + x678*xa*xb) +
     x5*((x6*x7 + x7*x8 + x678*x9)*xab + x678*xa*xb) +
     x6*((x7*x8 + x78*x9)*xab + x78*xa*xb);
```

```
D1 = x3sq*(x56*x7*x8 + x5*(x6*x79+x2*x8) + (x1278*x5 + x16*x8)*x9 +
   x6*x7*x9b + x59b*x8*xa + x589b*(x16*x2 + x1*x7 + x1267*xa) +
   (x27*x8 + x12678*x9)*xb) +
   2*x1*x3*(x589b*x7*xa + x8*x9b*xa + (x79*x8 + x7*x9)*xb) +
   2*x2*x3*(x56*x79*x8 + x5*(x6*x79 + x7*x9) + x6*x7*x9b + x589b*x7*xa +
   x8*x9b*xa + (x79*x8+x7*x9)*xb)+
   x2sq*(x3479b*x5*x6 + x56*(x7*x8 + x34*x89) + (x5*x78 + x6*x8)*x9 +
   (x1*x58 + x6*x7)*x9b + x34*x58b*xa + x589b*(x1*x347 + x7*xa) +
   x3458*x9*xab + (x34*x6 + x34567*x8 + x7*x9)*xb + x58*xa*xb) +
   2*x1*x2*(x3a*x589b*x7 + (x34*x5 + x3*x8)*x9 + (x34*x58b + x8*x9)*xa +
   x345*x9*xab + (x379*x8 + x7*x9)*xb + x58*xa*xb) +
   x1sq*(x589b*(x2*x347 + x34a*x7) + x34*x58*x9a + x2*x58*x9b +
   x3458*x9*xab + x589*x7*xb + (x34 + x58)*xa*xb);

D = D1 + C*x456789ab; // Massive diagram
```

A.3 $M63$, Fig. 1(c).

```
C = x8*(x6*x7 + x567*x9)*xb +
   (x1*x345689 + x2356*x789 + x4*x79 + x7*x89)*xa*xb +
   x2*((x3*x45 + x4*x6)*x789 + x35*(x6a*x789 + x78*x9) +
   x4*(x78*x9 + x789*xab) + (x56*x789 + x78*x9)*xb) +
   x4*(x6*(x7*x8a + x8*x9a) + x7*x8*x9ab + x67*x9*xa + x8*x9a*xb) +
   x1*(x345*(x69a*x8 + x69*xa) + x4569*x8*xb +
   x3*(x45*x689b + x69*xb) + x2*(x3*x4569a + x45*x69ab + x69a*xb)) +
   x5*(x67*(x8*x9a + x9*xa) + x7*(x6*x8a + x8*xb)) +
   x3*(x8*(x6*x7 + x67*x9) + (x6*x789 + x7*x89)*xa +
   (x6*x789 + x78*x9)*xb + x45*(x6*x789 + x7*x89b + x89*xb));

D1 = 2*x2*x4*(x8*x9a + x79*xa)*xb + 2*x1*x4*(x8*x9a + x9*xa)*xb +
   2*x1*x3*(x6b*(x5*x8 + x4*xa) + x4*(x69ab*x8 + x9*xa) +
   (x69a*x8 + x4a*x9)*xb) +
   x2sq*(x3*(x1*x4569a + x456a*x789 + x78*x9) +
   x45*(x1*x69ab + x6*x789 + x78*x9 + x789*xab) +
   (x1*x69a + x6a*x789 + x78*x9)*xb) +
   2*x1*x2*(x3*(x5*x689b + x4*x69ab + x4a*x89 + x69*x8b) +
   x45*(x69ab*x8 + x9*xa) + (x69a*x8 + x4a*x9)*xb + x34*xa*xb) +
   2*x3*x4*(x6*x7*x8a + x67*(x8*x9a + x9*xa) +
   (x8*x9 + x789*xa)*xb) +
   x1sq*(x2*(x3*x4569a + x45*x69ab) + x45*(x69ab*x8 + x69b*xa) +
   x3*(x45*x689b + x6*x8a + x8*x9 + x89*xa) +
   (x23*x69a + x6*x8a + x8*x9a + x9*xa)*xb) +
   2*x2*x3*((x5*x7 + x4b*x78)*x9 + x789*(x4*x6ab + x5*x6b + x6a*xb)) +
   x4sq*(x8*(x6*x79 + x7*x9) + (x6*x7 + x67*x89)*xa +
   x1*(x2689b*x3 + x28*x69ab + x69b*xa) +
   x2*(x36*x789 + x8*x9 + x7*x9ab + x89*xab) +
   (x79*x8 + x789*xa)*xb + x3*(x6*x789 + x7*x89b + x89*xb)) +
   x3sq*(x6*x7*x8ab + x1*(x45a*x689b + x69*x8b) +
   x2*(x1*x4569a + x456a*x789 + x78*x9) + x67*(x8*x9 + x89*xa) +
   (x69*x8 + x67*x9 + x789*xa)*xb + x45*(x6*x789 + x7*x89b + x89*xb));

D = D1 + C*x56789ab; // Massive diagram
```

A.4 *BH*, Fig. 1(d).

```
C = x189*xa*(x36*x45b + x45*xb)
   + (x189*x27 + x18*x9)*(x36*x45b + x45*xab + xa*xb);

D1 = x3sq*(x45b*(x189*x2 + x1*x79a) +
   x45*(x7a*x89 + x8*x9)) + 2*x1*x3*x4*x7a*xb +
   x34sq*(x7a*x89 + x8*x9)*xb +
   x1sq*(x45b*(x279a*x36 + x279*xa) + x279a*x45*xb) +
   x4sq*(x18*x36a*x9 + x189*(x27a*x36 + x7*xa + x2*xab) +
   x1*x79a*xb) + x189*x2sq*(x36*x45b + x45*xab + xa*xb) +
   2*x1*x9*(x45b*(x2*x36 + x3*xa) + x2*x45*xab +
   (x3*x4 + x24*xa)*xb) +
   2*x189*x2*(x4*xa*xb + x3*(x45b*xa + x4*xb));

D = D1 + C*x56789ab; // Massive diagram
```

References

1. Abi, B., et al.: Measurement of the positive muon anomalous magnetic moment to 0.46 ppm. Phys. Rev. Lett. **126**, 141801 (2021). (Muon g-2 Collaboration). https://doi.org/10.1103/PhysRevLett.126.141801
2. Aoyama, T., et al.: The anomalous magnetic moment of the muon in the standard model. Phys. Rep. **887**, 1–166 (2020). https://doi.org/10.1016/j.physrep.2020.07.006
3. Baikov, B.A., Chetyrkin, K.G.: Four loop massless propagators: an algebraic evaluation of all master integrals. Nucl. Phys. B **837**, 186–220 (2010)
4. Berntsen, J., Espelid, T.O., Genz, A.: Algorithm 698: DCUHRE-an adaptive multidimensional integration routine for a vector of integrals. ACM Trans. Math. Softw. **17**, 452–456 (1991)
5. Binoth, T., Heinrich, G.: Numerical evaluation of multi-loop integrals by sector decomposition. Nucl. Phys. B **680**, 375 (2004). hep-ph/0305234v1
6. Borowka, S., Heinrich, G., Jahn, S., Jones, S.P., Kerner, M., Schlenk, J.: A GPU compatible quasi-Monte Carlo integrator interfaced to pySecDec. Comput. Phys. Commun. **240**, 120–137 (2019). Preprint: arXiv:1811.11720v1 [hep-ph]. https://arxiv.org/abs/1811.11720. https://doi.org/10.1016/j.cpc.2019.02.015
7. Borowka, S., et al.: pySecDec: a toolbox for the numerical evaluation of multi-scale integrals. Comput. Phys. Commun. **222**, 313–326 (2018). arXiv:1703.09692 [hep-ph]. https://www.sciencedirect.com/science/article/pii/S0010465517303028. https://doi.org/10.1016/j.cpc.2017.09.015
8. Carter, J., Heinrich, G.: SecDec: a general program for sector decomposition. Comput. Phys. Commun. **182**, 1566–1581 (2011)
9. cgchannel: 28 May 2020. http://www.cgchannel.com/2020/05/review-amd-ryzen-threadripper-3990x/
10. Cranley, R., Patterson, T.N.L.: Randomization of number theoretic methods for multiple integration. SIAM J. Numer. Anal. **13**, 904–914 (1976)
11. De Ridder, L., Van Dooren, P.: An adaptive algorithm for numerical integration over an N-dimensional cube. J. Comput. Appl. Math. **2**(3), 207–210 (1976)
12. de Doncker, E., Almulihi, A., Yuasa, F.: High speed evaluation of loop integrals using lattice rules. J. Phys. Conf. Ser. (JPCS), IOP Ser. **1085**(052005) (2018). http://iopscience.iop.org/article/10.1088/1742-6596/1085/5/052005

13. de Doncker, E., Almulihi, A., Yuasa, F.: Transformed lattice rules for Feynman loop integrals. J. Phys. Conf. Ser. (JPCS) IOP Ser. **1136**(012002) (2018). https://doi.org/10.1088/1742-6596/1136/1/012002
14. de Doncker, E., et al.: Quadpack computation of Feynman loop integrals. J. Comput. Sci. (JoCS) **3**(3), 102–112 (2011). https://doi.org/10.1016/j.jocs.2011.06.003
15. de Doncker, E., Genz, A., Gupta, A., Zanny, R.: Tools for distributed adaptive multivariate integration on now's: ParInt1.0 release. In: Supercomputing 1998 (1998)
16. de Doncker, E., Kaugars, K., Cucos, L., Zanny, R.: Current status of the ParInt package for parallel multivariate integration. In: Proceedings of Computational Particle Physics Symposium (CPP 2001), pp. 110–119 (2001)
17. Okada, H., Atluri, S.N. (eds.): ICCES 2019. MMS, vol. 75. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-27053-7
18. de Doncker, E., Yuasa, F., Almulihi, A., Nakasato, N., Daisaka, H., Ishikawa, T.: Numerical multi-loop integration on heterogeneous many-core processors. J. Phys. Conf. Ser. (JPCS) **1525**(012002) (2019). https://doi.org/10.1088/1742-6596/1525/1/012002
19. de Doncker, E., Yuasa, F., Kato, K., Ishikawa, T., Kapenga, J., Olagbemi, O.: Regularization with numerical extrapolation for finite and UV-divergent multi-loop integrals. Comput. Phys. Commun. **224**, 164–185 (2018). https://doi.org/10.1016/j.cpc.2017.11.001
20. de Doncker, E., Yuasa, F., Olagbemi, O., Ishikawa, T.: Large scale automatic computations for Feynman diagrams with up to five loops. In: Gervasi, O., et al. (eds.) ICCSA 2020. LNCS, vol. 12253, pp. 145–162. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58814-4_11
21. Genz, A., Malik, A.: An adaptive algorithm for numerical integration over an n-dimensional rectangular region. J. Comput. Appl. Math. **6**, 295–302 (1980)
22. Hahn, T.: Cuba - a library for multidimensional numerical integration. Comput. Phys. Commun. **176**, 712–713 (2007). https://doi.org/10.1016/j.cpc.2007.03.006
23. Korobov, N.M.: The approximate computation of multiple integrals. Dokl. Akad. Nauk SSSR **124**, 1207–1210 (1959). (Russian)
24. Korobov, N.M.: Properties and calculation of optimal coefficients. Doklady Akademii Nauk SSSR **132**, 1009–1012 (1960). (Russ.). Eng. trans. Soviet Math. Doklady **1**, 696–700
25. L' Equyer, P., Munger, D.: Algorithm 958: lattice builder: a general software tool for constructing rank-1 lattice rules. ACM Trans. Math. Softw. **42**(2), 15:1–30 (2016)
26. Lee, R.N., Smirnov, A.V., Smirnov, V.A.: Master integrals for four-loop massless propagators up to weight twelve. Nucl. Phys. B **856**, 95–110 (2012)
27. Malcolm, M., Simpson, R.: Local versus global strategies for adaptive quadrature. ACM Trans. Math. Softw. **1**, 129–146 (1975)
28. MPI: http://www-unix.mcs.anl.gov/mpi/index.html
29. Niederreiter, H.: Existence of good lattice points in the sense of Hlawka. Monatshefte für Mathematik **86**, 203–219 (1978)
30. Nuyens, D., Cools, R.: Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing kernel Hilbert spaces. Math. Comp. **75**, 903–920 (2006)
31. Nuyens, D., Cools, R.: Fast component-by-component construction of rank-1 lattice rules with a non-prime number of points. J. Complex. **22**, 4–28 (2006)
32. Piessens, R., de Doncker, E., Überhuber, C.W., Kahaner, D.K.: QUADPACK, A Subroutine Package for Automatic Integration. Springer Series in Computational Mathematics, vol. 1. Springer, Heidelberg (1983). https://doi.org/10.1007/978-3-642-61786-7
33. Rice, J.R.: A metalgorithm for adaptive quadrature. J. Assoc. Comput. Mach. **22**, 61–82 (1975)

34. Ruijl, B., Ueda, T., Vermaseren, J.A.M.: Forcer, a FORM program for the parametric reduction of four-loop massless propagator diagrams. Comput. Phys. Commun. **253**(107198) (2020)
35. Shanks, D.: Non-linear transformations of divergent and slowly convergent sequences. J. Math. and Phys. **34**, 1–42 (1955)
36. Sidi, A.: Extension of a class of periodizing transformations for numerical integration. Math. Comp. **75**(253), 327–343 (2005)
37. Sloan, I., Joe, S.: Lattice Methods for Multiple Integration. Oxford University Press, Oxford (1994)
38. Smirnov, A.V., Tentyukov, M.: Four-loop massless propagators: an numerical evaluation of all master integrals. Nucl. Phys. B **837**, 40–49 (2010)
39. Sugihara, M.: Optimality of the double exponential formula - functional analysis approach. Numer. Math. **75**(3), 379–395 (1997)
40. Takahasi, H., Mori, M.: Double exponential formulas for numerical integration. Publ. Res. Inst. Math. Sci. **9**(3), 721–741 (1974)
41. Techpowerup. https://www.techpowerup.com/gpu-specs/quadro-gv100.c3066
42. Wynn, P.: On a device for computing the $e_m(s_n)$ transformation. Math. Tables Aids Comput. **10**, 91–96 (1956)