Computer Physics Communications 69 (1992) 173–181
North-Holland

Computer Physics Communications

Calculation of long traces of γ-matrices in the dimensional regularization scheme

Dirk Graudenz [1]

II. Institut für Theoretische Physik der Universität Hamburg, Luruper Chaussee 149, W-2000 Hamburg 50, Germany

Received 24 May 1991

We describe the program DTRACE that is capable of calculating products of traces of γ-matrices in d dimensions. DTRACE gives the result in a form suitable as input for algebraic manipulation programs.

PROGRAM SUMMARY

Title of program: DTRACE

Catalogue number: ACBX

Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland (see application form in this issue)

Licensing provisions: none

Computer: Micro VAX

Operating system: UNIX

Programming language used: C

Memory required to execute with typical data: 3000 kBytes

No. of bits in a word: 32

Number of lines in distributed program, including test data, etc.: 3000

Keywords: Dirac algebra, gamma matrices, symbolic calculation

Nature of physical problem
The calculation of the multi-particle cross-section involves the calculation of long traces of Dirac matrices. Even symbolic manipulation programs are not capable of doing this because of the large number of terms.

Method of solution
The terms of the trace of a γ-string are generated recursively.

Restrictions on the complexity of the problem
Applicability depends on the available computing power; traces of 18 γ-matrices can be calculated in a reasonable time. The length of the output depends on the number of free vectors.

Typical running time
Depends on the length of the trace, 30 minutes for 16 matrices.

[1] Supported by Studienstiftung des deutschen Volkes

LONG WRITE-UP

1. Introduction

For the calculation of cross-sections of processes involving fermions and multi-particle final states, the calculation of long traces of γ-matrices is necessary. The calculation of five-jet cross-sections and four-jet one-loop contributions involves traces of up to 16 γ-matrices leading to 2027025 terms that must be summed. On the tree-graph level, no regularization is needed, and in 4 dimensions, there are very efficient algorithms by Kahane and Chisholm that simplify the calculations considerably (see a recent review by Veltman [1] and references therein). These methods, however, do not work in d dimensions. Here it seems to be that using the general relations

$$\{\gamma^\mu, \gamma^\nu\} = 2g^{\mu\nu}, \tag{1.1}$$

$$g^\mu_\mu = d, \tag{1.2}$$

one has to anticommute the matrices in the trace and use the cyclic invariance to reduce the problem to the calculation of shorter traces. These formulae finally lead to the expression

$$\begin{aligned}&\mathrm{Tr}(\not a_1 \not a_2 \ldots \not a_{2n}) \\ &= 2\sum_{j=2}^{2n} (-1)^j (a_1 \cdot a_j) \\ &\quad \times \mathrm{Tr}\big(\not a_2 \not a_3 \ldots \hat{\not a}_j \ldots \not a_{2n}\big),\end{aligned} \tag{1.3}$$

where the factor under the $\hat{}$ has to be omitted.

Algebraic manipulation programs like REDUCE [2] and SMP [3] use eq. (1.3) and do this task very well up to a certain number of γ-matrices where the limitation is available memory and computer time. General traces of 16 and more γ-matrices in arbitrary dimensions d cannot be calculated by these programs in a reasonable time.

This paper describes the program DTRACE that can calculate these traces by using a limited amount of computer memory giving the result in a form that may be read in by SMP for further calculations. The principle of DTRACE is the following. The evaluated trace is a sum of products of a polynomial in the space–time dimension $d = 4 - 2\epsilon$ and a product of inner products of d-vectors. Here the polynomials have integer coefficients. These integer coefficients are calculated by using the algorithm given below.

The outline of the paper is as follows. In section 2, we describe the general method. Section 3 contains a description of the algorithm, section 4 briefly describes details of DTRACE and section 5 gives a simple example.

2. Calculation of traces

First we will describe the method if only one trace has to be calculated. Later we will generalize the method to a product of several traces.

Let $p_1, \ldots, p_m$ be a set of momenta and $\mu_1, \ldots, \mu_k$ a set of indices that are summed implicitly from 1 to d, the space–time dimension. Each of these indices has to appear exactly twice, so that the trace consists of $2n = m + 2k$ γ-matrices. We assume that m is even, otherwise the trace vanishes. For an arbitrary index we have a situation like

$$\mathrm{Tr}(\ldots\overbrace{\gamma^\mu \ldots \gamma_\mu}\ldots). \tag{2.1}$$

The bracket on the top of the trace indicates the implicit sum and will be called an *upper contraction*. During the calculation, the upper contractions are fixed since the positions of the summed indices do not change.

A well known formula [4] that can be derived from (1.3) states that

$$\begin{aligned}&\mathrm{Tr}(\not a_1 \not a_2 \ldots \not a_{2n}) \\ &= 4\sum \epsilon(i, j)(a_{i_1} \cdot a_{j_1}) \cdots (a_{i_n} \cdot a_{j_n}),\end{aligned} \tag{2.2}$$

where $(i_1, j_1), \ldots, (i_n, j_n)$ runs over all $(2n)!/2^n n!$ pairs such that

$$1 = i_1 < i_2 < \ldots < i_n < 2n \text{ and } i_k < j_k, \tag{2.3}$$

and $\epsilon(i, j)$ is the signature of this permutation.

If $\not b$ is γ^μ, then $a \cdot b := a^\mu$, if $\not a = \gamma^\mu$, $\not b = \gamma^\nu$, then $a \cdot b := g^{\mu\nu}$.

Every permutation will be symbolized by a *complete lower contraction* (i, j):

$$\mathrm{Tr}(\underbrace{\not{a}_{i_1}\ldots\not{a}_{i_2}}\ldots\underbrace{\not{a}_{j_1}\ldots\not{a}_{j_2}}). \tag{2.4}$$

It is easy to see that the sum over all permutations satisfying (2.3) is equivalent to a sum over all complete lower contractions. Of course, the sign of each permutation has to be determined separately.

Now, a typical term in the sum of $(2n)!/2^n n! = 105$ contributions is

$$\mathrm{Tr}(\not{p}_1\gamma^\mu\not{p}_2\,\gamma^\nu\,\not{p}_3\,\not{p}_4\,\gamma_\mu\gamma_\nu), \tag{2.5}$$

which is equal to

$$\begin{aligned}&-(p_1\cdot p_3)p_2^\mu p_{4\mu}g_\nu^\nu\\ &\quad=-(p_1\cdot p_3)(p_2\cdot p_4)d\ \mathrm{Tr}(\mathbf{1}),\end{aligned} \tag{2.6}$$

where $\mathrm{Tr}(\mathbf{1}) = 4$.

The inner product $p_1\cdot p_3$ is the result of one lower contraction. The product $p_2\cdot p_4$ comes in because of the lower contraction (γ^μ, p_2), the upper contraction (γ^μ, γ_μ) and the lower contraction (p_4, γ_μ). The factor d is the result of g_ν^ν, which comes from the lower contraction (γ^ν, γ_ν) and the upper contraction (γ^ν, γ_ν).

Of course, this is not the only contribution proportional to $(p_1\cdot p_3)(p_2\cdot p_4)$. If all these terms are summed, we obtain $(16 - 16\epsilon^2)(p_1\cdot p_3)(p_2\cdot p_4)$.

These relations hold in general: We may start with an external d-vector $\not{p}_1$ in the trace. Then we have to follow a continuous line of lower and upper contractions until we reach another external d-vector p_j. Clearly this will contribute a factor of $p_i\cdot p_j$. The other possibility is a closed loop of upper and lower contractions. Here, no d-vector may occur. Finally, these closed loops reduce to factors g_ν^ν each giving a factor of d. In addition we have to determine a sign.

In the example above, the sign is (-1), because the permutation

$$\begin{pmatrix}1&2&3&4&5&6&7&8\\1&5&2&3&4&8&6&7\end{pmatrix} \tag{2.7}$$

is odd.

3. The algorithm

Now the algorithm for the calculation of the traces may be formulated.

Given a distinct complete lower contraction, determine continuous lines connecting two external d-vectors p_i, p_j giving a factor $p_i\cdot p_j$. Determine the number c of closed loops of upper and lower contractions, resulting in a factor d^c. Determine the sign of the permutation. Now sum over all possible complete lower contractions of the trace.

The generalization to a product of several traces is straightforward. If there are summed indices μ in two different traces, there is an upper contraction between them. Again, according to the rule (2.3) we have to sum over all possible complete lower contractions, but we have to restrict ourselves to the case that the lower contractions do *not* connect different traces. The rules for the various factors are the same as above.

If the lower contractions are generated recursively, then there is a very simple rule to determine the sign of the permutation. We will describe the case of one trace, the case of a product of traces is then a straightforward generalization.

Obviously, all we need is a recursive algorithm generating all (i, j) that satisfy (2.3).

We define $L_0 := \{1, 2, \ldots, 2n\}$.

In the kth step we assume that $(i_1, j_1), \ldots, (i_{k-1}, j_{k-1})$ is already constructed.

Define $L_k := L_0 \setminus \{i_1, j_1, \ldots, i_{k-1}, j_{k-1}\}$,
$i_k := \min L_k$, $L_k' := L_k \setminus \{i_k\}$.

Now let h_k be the unique strongly isotonic map

$$h_k\colon \{1, 2, \ldots, 2(n-k)+1\} \to L_k'.$$

There are $2(n-k)+1$ elements left in L_k' corresponding to $2(n-k)+1$ different choices for $j_k = h_k(\alpha_k)$, $1 \le \alpha_k \le 2(n-k)+1$. We fix an α_k.

If $k \neq n$, we continue with step $k+1$.

This generates all complete lower contractions. The sign of the permutation is simply

$$\epsilon(i, j) = \prod_{k=1}^{n} (-1)^{\alpha_k + 1}. \tag{3.1}$$

4. DTRACE

The program DTRACE is written in C [5]. It consists of three parts:

(1) a parser [6] that reads a program in the file PROG formulated in a context-free language [7],
(2) a part that calculates the traces according to the instructions from a parameter file PAR using the algorithm described in section 3,
(3) a "pretty printer" that prints the result to an output file RES in a form suitable as input for SMP (in principle, it is possible to modify the pretty printer such that REDUCE input is generated).

DTRACE creates some more files, one contains information concerning the parsing process (e.g. a symbol table) and the internal form of the traces that were calculated, a temporary file used for storing some intermediate results and a file for error messages. All file names are declared at the beginning of PAR.

In PROG, vectors, indices, fermion lines and Lorentz scalars have to be declared. Then there follows a list of factors (called TERMs) that represent products of γ-matrices in different fermion lines, scalars, inner products, components of vectors, and so on.

The parameter file is simply a list of expressions $(t_1, \ldots, t_r)$, where t_i are TERMs defined in PROG. DTRACE then calculates the relevant traces of $t_1 \cdot t_2 \cdot \ldots \cdot t_r$.

In the Test Run section we give an SMP program SUM01 that sums all contributions leading to a result in the file SMPRES.

5. A simple example

We will calculate the trace for the cut vacuum diagram with massless fermions, given in fig. 1. Using the Feynman rules, we obtain

$$\begin{aligned}&\mathrm{Tr}\big(\not p_i\gamma_\nu(\not p_1-\not k)\gamma_\mu(-(\not p_2+\not k))\gamma^\nu\not p_2\gamma^\mu\big)\\&\quad=\mathrm{Tr}\Big(\gamma^\mu\not p_1\big(-\gamma_\nu\not p_1\gamma_\mu\not p_2\gamma^\nu-\gamma_\nu\not p_1\gamma_\mu\not k\gamma^\nu\\&\qquad+\gamma_\nu\not k\gamma_\mu\not p_2\gamma^\nu+\gamma_\nu\not k\gamma_\mu\not k\gamma^\nu\big)\not p_2\Big).\end{aligned}$$

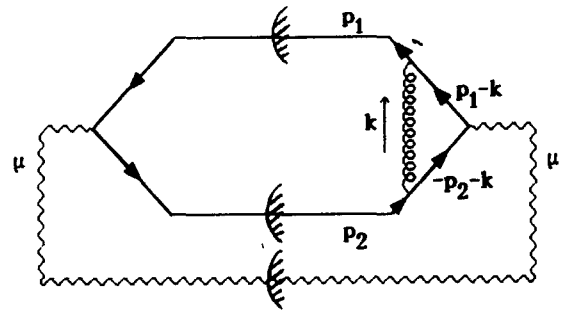

Fig. 1. A cut vacuum diagram.

We split this sum into four terms. Each of them is calculated separately. The PROG-file for this example is given in the Test Run section as PROG01. Comments are enclosed in [ ]'s.

The parameter file for the calculation is PAR01. The LOOP(1, 1, 1, 4) statement counts the symbolic variables $1 and $2 from 1 to 1 and 1 to 4, respectively, and replaces $1 and $2 by numerical values in the following term. Text that appears in between double quotes is copied to the output file. An exclamation mark in the file creates a new line in the output file.

The program SUM01 sums the terms from the output file RES. The final result is in the file SMPRES.

In a similar manner, arbitrary products of traces may be calculated. On a μVAX, the 15!! = 2027025 terms of a trace with 16 γ-matrices are calculated in 27 minutes. Traces of 18 γ-matrices need approximately 9 hours CPU-time. Another limit is the number f of free d-vectors in the trace. There will be $(f-1)!!$ terms in the output file, and $f = 10$ seems to be a reasonable limit $(9!! = 945)$.

6. Summary and conclusions

In this paper we have described DTRACE. By using this tool, very long traces of γ-matrices in the dimensional regularization scheme may be calculated. The algorithm is a straightforward implementation of the well known formula (2.2).

Acknowledgements

I would like to thank Prof. G. Kramer who brought this problem to my attention and M. Ernst from R01/DESY for solving problems concerning the μVAX.

References

[1] M. Veltman, Gammatrica, UM-TH-88-17, and references therein
[2] A.C. Hearn, REDUCE User's Manual, Version 3.2., Rand Publication CP78 (Rev. 4/85).
[3] S. Wolfram, SMP User's Manual.
[4] C. Itzykson and J.-B. Zuber, Quantum Field Theory, (McGraw-Hill, Singapore, 1985), Appendix A-2.
[5] B.W. Kernighan and D.M. Ritchie, The C Programming Language (Prentice-Hall, Englewood Cliffs, NJ, 1978).
[6] N. Wirth, Compilerbau (Teubner, Stuttgart, 1981).
A.V. Aho, R. Sethi and J.D. Ullman, COMPILERS: Principles, Techniques and Tools (Bell Telephone Laboratories, 1986).
[7] J.E. Hopcroft and J.D. Ullman, Introduction to automata theory, languages and computation (Addison-Wesley, Reading, MA, 1979).

TEST RUN

PROG01

```
BEGIN

[ Declarations ]

SCALAR minus1;

VECTOR p1, p2;
VECTOR k;
INDEX  my, ny;
FLINE  l1;

[------------------------------------------------------------------------]

[ Propagators ]

TERM PR_1 {
   GSTRING(l1, p1);
};

TERM PR_2 {
   GSTRING(l1, p2);
};

[------------------------------------------------------------------------]

TERM tleft {
   GSTRING(l1, my);
};

TERM tright1 {
   FACTOR minus1;
   GSTRING(l1, ny, p1, my, p2, ny);
};

TERM tright2 {
   FACTOR minus1;
   GSTRING(l1, ny, p1, my, k, ny);
};

TERM tright3 {
   GSTRING(l1, ny, k, my, p2, ny);
};

TERM tright4 {
   GSTRING(l1, ny, k, my, k, ny);
};

[------------------------------------------------------------------------]

END

[------------------------------------------------------------------------]
```

PAR01

```
[ Parameter file for Example 1 ]

[------------------------------------------------------------------------]

[ Filenames of external files ]

"prog01"
"res01"
"pinfo01"
"fitemp01"

[------------------------------------------------------------------------]

BEGIN

[ determines which information is printed to the file pinfo01 ]

?  2
?  4
? -5
?  6
?  7
?  8

[------------------------------------------------------------------------]

"/*==> Example 1... <==*/"!

[ Sum over the four contributions, the $2 is replaced by the numbers 1..4,
  respectively                                                          ]

"proc10"!
LOOP(1, 1, 1, 4)
(tleft, PR_1, tright$2, PR_2)
"proc11"!

[------------------------------------------------------------------------]

!"/*--> End of File <--*/"!

END

[------------------------------------------------------------------------]
```

RES01

```
proc10

/*--------------------------------------------------------------------*/
proc00;
fact1:minus1;
fact2:1*FF;
fact3[1]:                                                          \
1*h[(p1.p1)]*h[(p2.p2)];
fact4[1]:                                                          \
f1[(-8)+6*DD+(-1)*DD^2];
fact3[2]:                                                          \
1*h[(p1.p2)]*h[(p1.p2)];
fact4[2]:                                                          \
f1[16+(-10)*DD+1*DD^2];
fact3[3]:                                                          \
1*h[(p1.p2)]*h[(p1.p2)];
fact4[3]:                                                          \
f1[(-8)+6*DD+(-1)*DD^2];
nos:3;
dol1:1;
dol2:1;
proc01;
/*--------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
```

SUM01

```
proc00;
fact1:minus1;
fact2:1*FF;
fact3[1]:                                                    \
1*h[(p1.p1)]*h[(k.p2)];
fact4[1]:                                                    \
f1[(-8)+6*DD+(-1)*DD^2];
fact3[2]:                                                    \
1*h[(p1.k)]*h[(p1.p2)];
fact4[2]:                                                    \
f1[16+(-10)*DD+1*DD^2];
fact3[3]:                                                    \
1*h[(p1.p2)]*h[(p1.k)];
fact4[3]:                                                    \
f1[(-8)+6*DD+(-1)*DD^2];
nos:3;
dol1:1;
dol2:2;
proc01;
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
proc00;
fact1:1;
fact2:1*FF;
fact3[1]:                                                    \
1*h[(p1.k)]*h[(p2.p2)];
fact4[1]:                                                    \
f1[(-8)+6*DD+(-1)*DD^2];
fact3[2]:                                                    \
1*h[(p1.p2)]*h[(k.p2)];
fact4[2]:                                                    \
f1[16+(-10)*DD+1*DD^2];
fact3[3]:                                                    \
1*h[(p1.p2)]*h[(k.p2)];
fact4[3]:                                                    \
f1[(-8)+6*DD+(-1)*DD^2];
nos:3;
dol1:1;
dol2:3;
proc01;
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
proc00;
fact1:1;
fact2:1*FF;
fact3[1]:                                                    \
1*h[(p1.k)]*h[(k.p2)];
fact4[1]:                                                    \
f1[(-8)+6*DD+(-1)*DD^2];
fact3[2]:                                                    \
1*h[(p1.k)]*h[(k.p2)];
fact4[2]:                                                    \
f1[16+(-10)*DD+1*DD^2];
fact3[3]:                                                    \
1*h[(p1.p2)]*h[(k.k)];
fact4[3]:                                                    \
f1[(-8)+6*DD+(-1)*DD^2];
nos:3;
dol1:1;
dol2:4;
proc01;
/*-------------------------------------------------------------------------*/
proc11                                                       \


/*--> End of File <--*/                                      \
```

```
/* Sum up all contributions */

DD:4-2*eps;   /* dimension of space-time */
FF:4;         /* trace of 1              */
minus1:-1;

h[$x]:$x;
f1[$x]:$x;

/* massless particles */
p1.p1:0;
p2.p2:0;

/* relativistic invariants */
p1.p2:s12/2;
p2.p1:s12/2;

Run["rm smpres"];

proc10::Proc[ Lpr["(proc10)"];                                          \
      sum:0;                                                    \
    ];

proc11::Proc[ Lpr["(proc11)"];                                          \
      Put[sum,"smpres"];                                        \
    ];

proc00::Proc[ Lpr["(proc00)"];                                          \
    ];

proc01::Proc[ Lpr["(proc01)"];                                          \
      Put[nos];                                                 \
      For[ ic:1, ic<=nos, Inc[ic],                              \
   Put[ic];                                             \
   a:Ex[fact1*fact2*fact3[ic]*fact4[ic]];               \
   sum:sum+a;                                           \
 ];                                                     \
    ];

Lpr["Start reading.."];
<"res01";
```

SMPRES

```
sum : -8(eps*s12^2) + -16(s12*k.p2) + 16(s12*p1.k) + -32(k.p2*p1.k)\
          + 8(eps*s12*k.k) + 16(eps*s12*k.p2) + -16(eps*s12*p1.k)\
          + 32(eps*k.p2*p1.k) + -8(eps^2*s12*k.k) + 8(s12^2)
```