Eur. Phys. J. C (2015) 75:27
DOI 10.1140/epjc/s10052-014-3249-9

THE EUROPEAN PHYSICAL JOURNAL C

Special Article - Tools for Experiment and Theory

CHIRON: a package for ChPT numerical results at two loops

Johan Bijnens[a]

Department of Astronomy and Theoretical Physics, Lund University, Sölvegatan 14A, 223-62 Lund, Sweden

Received: 3 December 2014 / Accepted: 22 December 2014 / Published online: 24 January 2015

Abstract This document describes the package CHIRON which includes two libraries, `chiron` itself and `jbnumlib`. `chiron` is a set of routines useful for two-loop numerical results in chiral perturbation theory (ChPT). It includes programs for the needed one- and two-loop integrals as well as routines to deal with the ChPT parameters. The present version includes everything needed for the masses, decay constants and quark-antiquark vacuum-expectation-values. An added routine calculates consistent values for the masses and decay constants when the pion and kaon masses are varied. In addition a number of finite volume results are included: one-loop tadpole integrals, two-loop sunset integrals and the results for masses and decay constants. The numerical routine library `jbnumlib` contains the numerical routines used in `chiron`. Many are to a large extent simple C++ versions of routines in the CERNLIB numerical library. Notable exceptions are the dilogarithm and the Jacobi theta function implementations. This paper describes what is included in CHIRON v0.50.

Contents

1 Introduction

Chiral perturbation theory (ChPT) is the low-energy effective field theory of QCD. It was introduced by Weinberg, Gasser and Leutwyler [1–3] and the present state of the art are calculations performed at two-loop level. A review is [4] but many more exists. The long term goal of this project is to make available all these calculations with a consistent interface in C++. Many of the original programs were written in FORTRAN77 and are available on request from the authors, but they are not always consistent in the interfaces and the

[a] e-mail: bijnens@thep.lu.se

use of common blocks for moving parameters around has occasionally lead to difficult to find errors.

A general knowledge of C++ is assumed throughout this paper. The routines are at present not guaranteed to be thread-safe, some global variables inside the various files are used for the loop functions and integration routines. These are however never used for setting outside the files, there are always functions provided for this. It is recommended to always use these. The routines return double precision types if not indicated directly.

Kheiron, $X\varepsilon\iota\rho\omega\nu$, or Chiron, was the eldest and wisest of the Centaurs, half-horse men of Greek mythology. His name is derived from the Greek word for hand (Kheir) which also formed the basis of the word chiral which is why his name was chosen for this package [5].

The license chosen is the General Public License v2 or later from the Free Software Foundation [6].

The `chiron` routines have mainly been tested against the FORTRAN codes of the original publications. These were in turn implemented in at least two independent versions originally. The `jbnumlib` routines have their output compared with the original CERNLIB routines in case they were simple translations to C++. In the other cases, the tests are described in the relevant sections.

This paper describes what is included in CHIRON v0.50. The package itself is available from [7]. The included files and how to install it is described in Sect. 2. The numerical routines included in `jbnumlib` are described in Sect. 3. Some short comments on ChPT notation are in Sect. 4. The main part describing the contents of `chiron` are Sects. 5 to 8. Section 5 contains the objects implemented to deal with input data for the ChPT calculations. A large part of the work is in implementing the relavant loop integrals, especially those at finite volume. This is the content of Sect. 6. The simplest quantities are masses, decay constants and vacuum-expectation-values. The functions for these are discussed in Sect. 7 and the finite volume extensions for masses and decay constants in Sect. 8. Some comments about errors, some warnings about the use of the routines and definitions in ChPT as well as a number of planned/possible extensions are discussed in Sect. 9. A short summary is given in the final section.

2 Files and setup

The package is delivered as a gzipped tarred file (`chironvvvv.tar.gz`), where vvvv is version information. Untarring it creates a directory `chironvvvv`, which is referred to as the root directory below.

The package has a number of subdirectories when delivered. The root directory contains a Makefile and files COPYING, INSTRUCTIONS and GUIDELINES. The main instructions to produce CHIRON are to do first "make libjbnumlib.a" to produce the numerical library and then "make libchiron.a" to produce the main library or simply "make" to do both. The latter also puts the newly produced library versions in the subdirectory `lib`. To install, put the two library files and the content of the `include` directory somewhere where they can be linked to and included. Linking should be indicated on the compile line with the options "-lchiron -ljbnumlib".

The subdirectory `doc` contains this manuscript, a filelist and possibly more files in future versions. The subdirectory `src` contains the source files and `include` the various header files. The subdirectory `test` contains a number of testing programs where the names "testyyy.cc" indicates a program testing the code in the source file "yyy.cc". A testing program can be compiled using "make testyyy" in the root directory. The program "a.out" should then produce the output as shown in the file "testyyy.dat" in the subdirectory `testoutputs`. The test subdirectory contains in addition the file LiCiBE14.dat with the latest determination of the LECs [8].

3 `jbnumlib`

The functions in this section are included to make the program collection self-contained. They are mainly implementations of well known programs in C++ and in particular many of the routines are a port to C++ from the CERNLIB [9] FORTRAN routines. Some, as mentioned in the respective texts, are fully original. The definitions are all in `jbnumlib.h` and contained in `libjbnumlib.a`. The implementations are in the files mentioned in each subsection below. In order to avoid conflict with other implementations all routines in this section have names starting with `jb`. The exact interface is best checked by looking in the include file `jbnumlib.h`.

3.1 Special functions

3.1.1 Dilogarithm or $\mathrm{Li}_2(x)$*:* `jbdli2(x)`

The way the vertex integrals are implemented requires a Spence or Li_2 function which returns complex values for all possible complex inputs. The routine implemented uses the algorithm given in [10], Appendix A up to Bernouilly number B_{28}. Defined in `jbdli2.cc`. For real numbers the output has been compared to that of the CERNLIB routine `DDILOG`. It has also been checked that the function satisfies a number of the relations between values with different arguments that were not used in its evaluation.

3.1.2 Bessel functions

The modified Bessel functions I_0, I_1, K_0, K_1 with real arguments are available as `jbdbesi0`, `jbdbesi1`, `jbdbesk0` and `jbdbesk1`. These are implemented in `jbdbesio` which is a simple port to C++ of the CERNLIB routines `dbesi0`,.... In addition the modified Bessel functions K_2, K_3, K_4 are available as `jbdbesk2`, `jbdbesk3` and `jbdbesk4`. These are evaluated using the recursion relations from K_0 and K_1.

3.1.3 Theta functions

The functions defined are related to the Jacobi theta functions. `jbdtheta30(q)` returns the function

$$\theta_{30}(q) = 1 + 2 \sum_{n=1,\infty} q^{(n^2)} . \tag{1}$$

It uses the idea behind the CERNLIB routines `DTHETA`. For small q it simply sums the series (1) and for larger q it uses the modular invariance and a series in the changed variable instead. The accuracy has been checked by running both series too much higher orders and comparing the two results. Implemented in `jbdtheta30.cc`. A function without the 1 which is needed to keep accuracy for small q is available as `jbdtheta30m1`. Implemented in `jbdtheta30m1.cc`.

`jbdtheta32(q)` returns the function

$$\theta_{32}(q) = 2 \sum_{n=1,\infty} n^2 q^{(n^2)} . \tag{2}$$

For small q it simply sums the series (2) and for larger q it uses the modular invariance and the derivative of the series in the changed variable The accuracy has been checked by running both series too much higher orders and comparing the two results. Defined in `jbdtheta32.cc`. A function with n^4 multiplying $q^{(n^2)}$ is available as `jbdtheta34` and implemented in `jbdtheta34.cc`.

3.1.4 Higher dimensional theta functions

There are higher dimensional generalizations of the theta functions. These satisfy a more general modular invariance which can be used to get much faster convergence series. The functions defined below use some of these possible optimizations.

The basic function is the 2-dimensional generalization

$$\theta_0^{(2d)}(\alpha, \beta, \gamma) = \sum_{n_1,n_2=-\infty,\infty} e^{-\alpha n_1^2 - \beta n_2^2 - \gamma (n_1 - n_2)^2} . \tag{3}$$

This function is implemented as `jbdtheta2d0` with arguments α, β, γ in `jbdtheta2d0.cc`. In addition the function with the one removed is also available as `jbdtheta2d0m1` implemented in `jbdtheta2d0m1.cc`.

The function with the exponential multiplied by n_1^2 is called `jbdtheta2d02` and implemented in `jbdtheta2d02.cc`.

3.2 Integration routines

An adaptive gaussian quadrature routine `jbdgauss` and an adaptive integration routine `jbdcauch`, that integrates symmetrically around a singularity, are included. These are ports of the CERNLIB routines `dgauss` and `dcauch` respectively. Implementation is in `jbdgauss.cc` and `jbdcauch.cc`. The complex equivalent is `jbwgauss` in `jbwgauss.cc`.

The higher dimensional integration CERNLIB routine `DADMUL` based on [11] has been ported to `C++` and a simple interface for two and three-dimensional integration implemented as `jbdad2` and `jbdad3`. The code can be found in `jbdadmul.cc`.

4 ChPT notation

The notation used in ChPT is not fully unique. The notation used in this program collection is the main one used by the author and his collaborators. The main point to be observed is that `chiron` uses a normalization for the decay constants with $F_\pi \approx 92$ MeV.

For the low-energy-constants, we use the conventions of [3,12,13] with dimensionless renormalized couplings L_i^r and C_i^r.

The lowest order couplings are denoted by F_0 and B_0. The quark masses are $\hat{m} = m_u = m_d$, note that we work in the isospin limit, and m_s. The lowest order masses[1] are given by

$$m_{\pi\,0}^2 = 2B_0\hat{m}\,, \quad m_{K\,0}^2 = B_0\left(\hat{m} + m_s\right),$$
$$m_{\eta\,0}^2 = \frac{B_0}{3}\left(2\hat{m} + 4m_s\right). \tag{4}$$

5 Data structures

This section discusses the structures available for dealing with masses, F_π, the L_i^r and C_i^r, and the subtraction constant μ. Note that μ is present in all three data structures and the user should make sure that their use is consistent. The default-value mechanism in `C++` has been used to define the values when the constructors are called with less than the full data needed.

[1] Note that the programs use an internal convention where `mhat`$= 2B_0\hat{m}$ and `mstrange`$= 2B_0 m_s$.

These data structures are implemented as classes.

Note that we assume dimensional units in GeV, but if all dimensional inputs are scaled accordingly the routines give the correct answer. However, typical precisions set are assuming ChPT applications with dimensional units in powers of GeV.

5.1 physmass: Masses, F_π, μ

The physical masses are defined in a class `physmass` defined in `inputs.h` and implemented in `inputs.cc`.

Private data members: `mpi, mk, meta, fpi, mu`.

Typical declaration: `physmass mass1(0.135, 0.495,0.548,0.0922,0.77);` The numbers given above are also the defaults.

These are the physical pion, kaon and eta mass, m_π, m_K, m_η the physical pion decay constant, F_π, and the subtraction point μ. The default constructor puts them all at some reasonable values but they can be created with any number of the inputs specified, starting from the left. In addition there are functions `void setmpi(double)` etc., defined that set one of the values only. These use the same default values.

The values can be obtained from the function `void out( mpi, mk, meta, fpi, mu)` that returns all of them via referenced doubles. Functions `double getmpi(void)` etc. are defined that return one of the values.

The output/input stream format is defined as well so `cout << mass1` and `cin >> mass1` make sense. The input stream should have the same format as the output stream produces. This works for all streams, not just the standard `cout` and `cin`.

A test for equality is defined which checks that all data members agree to 7 significant digits. So expressions like `if(mass1 == mass2)` can be used. This is relative precision, so it is assumed no mass is zero here. The reason for not using exact equality is that calculated masses might not be exactly the same using double precision variables.

5.2 Classes for the NLO LECS: `Li`

The class for dealing with the next-to-leading-order (NLO) low-energy constants (LECs) defined in [3] is named `Li`. This is implemented in `Li.cc` and defined in `Li.h`.

The `Li` class has 13 double precision variables to store the LECs $L_1^r, \ldots, L_{10}^r$, H_1^r, H_2^r and the subtraction scale μ. It also contains a string with a name for the set of constants. The LECs default to zero, the scale to 0.77 and the name to "nameless Li." When the LECs are referred to with numbers, 11,12 correspond to H_1^r, H_2^r respectively.

Typical declarations are:

```
Li Li1;  Li Lifitall(0.88e-3,0.61e-3,-3.
04e-3,0.75e-3,0.58e-3,0.29e-3,
-0.11e-3,-0.18e-3,5.93e-3,0.,0.,0.,0.77,
"fit All");
```

Operations defined on the `Li`: overloaded operators are defined such that sets of `Li` can be added or subtracted and multiplied by a number (double). The output/input stream format is defined as well so `cout << Li1` and `cin >> Li1` make sense. The input stream should have the same format as the output stream produces.

Member functions that can be used to set the parameters are `setli` which takes an integer and a double (in either order) as argument to set the corresponding LEC to the double, `setmu` which sets the scale[2] and `setname` which changes the name of the set of L_i^r.

Output member functions exist to obtain a single LEC, `out(int)`, or the 10 L_i^r, the 10 L_i^r and μ, the 12 LECs, the 12 LECs and μ, and the 12 LECs, μ and the name. These are all called out and return the results via a reference to 10, 11, 12, 13 double or 13 double and a string variable.

The member function `changescale` changes the scale μ and changes the L_i^r and H_i^r according to the scale dependence as obtained first in [3].

There are also three functions defined that return a set of random NLO LECs. These are `Lirandom` which gives each LEC a random value between $\pm 1/(16\pi^2)$. `LirandomlargeNc` does the same but leaves L_4^r, L_6^r and L_7^r zero. Finally, `LirandomlargeNc2` does the same but L_4^r, L_6^r and L_7^r get a random value between $\pm(1/3)/(16\pi^2)$. Note that $1/(16\pi^2) \approx 0.0063$ so the ranges include the values of the fitted L_i^r. The random numbers are generated using the system generator `rand()` so we recommend initializing using something like `srand(time (0))`. These latter functions were used in the random walks in the L_i^r in [14].

5.3 Classes for the NNLO LECS: `Ci`

The class for dealing with the next-to-next-to-leading-order (NNLO) low-energy constants (LECs) defined in [12] is named `Ci`. This is implemented in `Ci.cc` and defined in `Ci.h`. Note that this set of routines uses the convention where the C_i^r are dimensionless. The parameters in the Lagrangian have dimension mass$^{(-2)}$ but the definition of the subtracted C_i^r in [13] is dimensionless. Going from one-convention to the other is with the appropriate power[3] of F_π.

The `Ci` class has as private members a double precision array `Cr[95]`, which holds the C_i^r, $i = 1, 94$ in `Cr[i]`, the scale mu and string for the name. Defaults are zero for all the C_i^r, 0.77 GeV for the scale and "nameless Ci" for the name.

[2] This sets the scale simply, it does *not* change the numerical values of the LECs.

[3] The definition is with the chiral limit value F_0 but the difference is higher order.

Constructors are provided with as input a double array Cr[95], the scale and a name or a scale and a name or a scale only or no input. Typical declarations are:
`Ci Ci1,Ci2(1.0),Ci3(Crr,0.8,"a nice set")`
where `Crr` is defined as `double Crr[95]`.

An additional constructor is provided that has as input the resonance parameters where the resonance model is the simple version described in Sect. 5 of [15].

Operations defined on the `Ci` are: overloaded operators are defined such that sets of `Ci` can be added or subtracted and multiplied by a number (double). The output/input stream format is defined as well so `cout << Ci1` and `cin >> Ci1` make sense. The input stream should have the same format as the output stream produces.

Member functions that can be used to set the parameters are `setci` which takes an integer and a double (in either order) as argument to set the corresponding LEC to the double, `setmu` which sets the scale[4] and `setname` which changes the name of the set of C_i^r.

Output member functions exist to obtain a single LEC, `out(int)`, or the `Cr[95]`, or `Cr[95]` and the scale, or `Cr[95]`, scale and name. These are all called out and return the results via references to the string and scale and the array.

The member functions `changescale(double,Li)` and `changescale(Li,double)` change the scale μ and changes the L_i^r, H_i^r and C_i^r according to the scale dependence as obtained first in [13]. Note that the `Li` set here has also the scale and values changed accordingly, not only the C_i^r.

There are also three functions defined that return a set of random NLO LECs. These are `Cirandom` which gives each LEC a random value between $\pm 1/(16\pi^2)^2$. `CirandomlargeNc` does the same but leaves the large-N_c suppressed constants zero. Finally, `CirandomlargeNc2` does the same but the large-N_c suppressed constants get a value between $\pm(1/3)/(16\pi^2)^2$. The random numbers are generated using the system generator `rand()` so we recommend initializing using something like `srand(time(0))`. Typically these values of the C_i^r are somewhat on the large side when fitting data.

6 Loop integrals

Most of the integrals used have been treated in many places. I refer only to the papers where our particular notation has been defined and/or the method used to evaluate them was developed. When comparing with other packages, keep in mind the differences in subtraction and/or differences in defining the integrals.

[4] This sets the scale simply, it does *not* change the numerical values of the LECs.

6.1 One-loop integrals

6.1.1 Tadpoles

These integrals have been defined in [16] and correspond to the finite parts of the integral

$$A(n,m^2) = \frac{1}{i}\int \frac{d^dp}{(2\pi)^d}\frac{1}{(p^2-m^2)^n}\,. \tag{5}$$

After the subtraction and renormalization as usual in ChPT, we are left with the finite four-dimensional part $\bar{A}(n,m^2)$ which is implemented as `Ab(n,msq,mu2)` and the $n=1,2,3$ as `Ab`, `Bb`, `Cb` respectively with arguments `msq`, `mu2`. These functions are defined in `oneloopintegrals.h` and implemented in `oneloopintegrals.cc`.

6.1.2 Bubble integrals

These have been defined in [16,17]

$$\begin{aligned}
B(m_1^2,m_2^2,p^2) &= \frac{1}{i}\int \frac{d^dq}{(2\pi)^d}\frac{1}{(q^2-m_1^2)((q-p)^2-m_2^2)}\,,\\
B_\mu(m_1^2,m_2^2,p^2) &= \frac{1}{i}\int \frac{d^dq}{(2\pi)^d}\frac{q_\mu}{(q^2-m_1^2)((q-p)^2-m_2^2)}\\
&= p_\mu B_1(m_1^2,m_2^2,p^2)\,,\\
B_{\mu\nu}(m_1^2,m_2^2,p^2) &= \frac{1}{i}\int \frac{d^dq}{(2\pi)^d}\frac{q_\mu q_\nu}{(q^2-m_1^2)((q-p)^2-m_2^2)}\\
&= p_\mu p_\nu B_{21}(m_1^2,m_2^2,p^2)\\
&\quad + g_{\mu\nu}B_{22}(m_1^2,m_2^2,p^2)\,,\\
B_{\mu\nu\alpha}(m_1^2,m_2^2,p^2) &= \frac{1}{i}\int \frac{d^dq}{(2\pi)^d}\frac{q_\mu q_\nu q_\alpha}{(q^2-m_1^2)((q-p)^2-m_2^2)}\\
&= p_\mu p_\nu p_\alpha B_{31}(m_1^2,m_2^2,p^2)+(p_\mu g_{\nu\alpha}+p_\nu g_{\mu\alpha}+p_\alpha g_{\mu\nu})\\
&\quad \times B_{32}(m_1^2,m_2^2,p^2)\,.
\end{aligned} \tag{6}$$

Again one needs to do the subtraction and renormalization with ChPT convention. The analytical values can be obtained using the methods of [10] for the B integral and the others can be reduced to it using the methods of [18]. All functions have been implemented via a method that does the integration over the Feynman parameter x numerically. These have real arguments `m1sq,m2sq,psq,mu2` and are called `Bbnum,B1bnum,B21bnum,B22bnum,B31bnum,B32bnum` and return a complex value. The analytical evaluation has been implemented in `Bb,B1b, B21b,B22b` with the same arguments and a complex return value. The simpler analytical expression for the case of the two masses equal has been implemented analytically for `Bb` and `B22b` called with argument `msq,psq,mu2`

For the cases with numerical integrations, the precision can be set using
`setprecisiononeloopintegrals(double)` and obtained by
`getprecisiononeloopintegrals(void)`.

All functions are defined in `oneloopintegrals.h` and implemented in `oneloopintegrals.cc`.

6.2 Sunset integrals

These give the sunsetintegral loop integral functions H^F, H_1^F, H_{21}^F defined in App. C of [16]. The definition in finite volume is given in (9) below. H_{31}^F is the function multiplying the $p_\mu p_\nu p_\rho$ part of the integral with $r_\mu r_\nu r_\rho$. These exist in a real version valid below threshold and a complex version valid everywhere. The method used is derived in [16].

The derivative w.r.t. p^2 is included for the real version of $\overline{H}, \overline{H}_1, \overline{H}_{21}$. Functions defined in `sunsetintegrals.h` and implemented in `sunsetintegrals.cc`.

Input arguments are real and are $m_1^2, m_2^2, m_3^2, p^2, \mu^2$. Naming conventions are `hh,hh1, hh21,hh31` for the real versions valid below threshold and the complex versions valid everywhere `zhh,zhh1,zhh21,zhh31`. The real versions are normally faster when applicable. In addition, wave-function renormalization requires some derivatives w.r.t. the external momentum p^2. These are encoded in `hhd,hh1d,hh21d` and `zhhd,zhh1d,zhh21d` for the real and complex case respectively.

The precision of the numerical integrations can be set using
`setprecisionsunsetintegrals(double)` and obtained by
`getprecisionsunsetintegrals(void)`.

These functions are defined in `sunsetintegrals.h` and implemented in `sunsetintegrals.cc`.

6.3 One-loop finite volume integrals

The methods used for these are derived in detail in [19], references to earlier literature can be found there. The integrals used here are given in the Minkowski conventions as defined in [20]. All of the integrals are available with two different methods, one using a summation over Bessel functions and the other an integral over a Jacobi theta function. The versions included at present are using periodic boundary conditions, all three spatial sizes of the same length L and the time direction of infinite extent.

6.3.1 Tadpoles

The tadpole integrals A and $A_{\mu\nu}$ are defined as

$$\left\{A(m^2), A_{\mu\nu}(m^2)\right\} = \frac{1}{i}\int_V \frac{d^d r}{(2\pi)^d}\frac{\left\{1, r_\mu r_\nu\right\}}{(r^2-m^2)}\,. \tag{7}$$

The B tadpole integrals are the same but with a doubled propagator.

The subscript V on the integral indicates that the integral is a discrete sum over the three spatial components and an integral over the remainder. At finite volume, there are more Lorentz-structures possible. The tensor $t_{\mu\nu}$, the spatial part of the Minkowski metric $g_{\mu\nu}$, is needed for these. The functions for $A_{\mu\nu}$ are

$$A_{\mu\nu}(m^2) = g_{\mu\nu}A_{22}(m^2) + t_{\mu\nu}A_{23}(m^2)\,. \tag{8}$$

In infinite volume A_{22} is related to A and A_{23} vanishes. We denote the finite volume part by a superscript V and one should remember that for the full integrals, the infinite volume results of Sect. 6.1.1 need to be added.

The functions are defined as `AbVt(msq,L)`, `BbVt(msq,L)`, `AbVb(msq,L)`, `BbVb(msq,L)`. The last letter indicates whether they are computed with the theta function or Bessel function method. The results were checked by comparing against each other and by comparing with the independent Bessel function implementation done in [21].

The functions `A22bVt(msq,L)`, `A22bVt(msq,L)`, and `A23bVb(msq,L)`,`A23bVb(msq,L)` are available as well.
`setprecisionfinitevolumeoneloopt(Abacc,Bbacc,printout)` and `setprecisionfinitevolumeoneloopt(maxsum,Bbacc,printout)` set the precision. The last variable printout is a logical variable which can be set to true or false, default is false. The first and second argument give the (mainly absolute) precision of the numerical integration for the tadpole and bubble integral numerical integrations. maxsum indicates how far the sum over Bessel function is taken. Maximum at present is 400.

These functions are defined in `finitevolumeoneloopintegrals.h` and implemented in `finitevolumeoneloopintegrals.cc`.

6.3.2 Bubble integrals

Not implemented in this version.

6.4 Sunset finite volume integrals

Sunset integrals are defined as

$$\left\{H, H_\mu, H_{\mu\nu}\right\} = \frac{1}{i^2}\int_V \frac{d^d r}{(2\pi)^d}\frac{d^d 1}{(2\pi)^d} \times \frac{\left\{1, r_\mu, r_\mu r_\nu\right\}}{\left(r^2-m_1^2\right)\left(s^2-m_2^2\right)\left((r+s-p)^2-m_3^2\right)}. \tag{9}$$

The subscript V indicates that the spatial dimensions are a discrete sum rather than an integral. The conventions correspond to those in infinite volume of [16]. Integrals with the other momentum s in the numerator are related using the trick shown in [16] which remains valid at finite volume in the cms frame [19].

In the cms frame we define the functions[5]

$$\begin{aligned} H_\mu &= p_\mu H_1 \\ H_{\mu\nu} &= p_\mu p_\nu H_{21} + g_{\mu\nu} H_{22} + t_{\mu\nu} H_{27}\,. \end{aligned} \tag{10}$$

The arguments of all functions in the cms frame are $(m_1^2, m_2^2, m_3^2, p^2)$. These functions satisfy the relations, valid in finite volume [19],

$$\begin{aligned} &H_1(m_1^2, m_2^2, m_3^2, p^2) + H_1(m_2^2, m_3^2, m_1^2, p^2) \\ &\quad + H_1(m_3^2, m_1^2, m_2^2, p^2) = H(m_1^2, m_2^2, m_3^2, p^2)\,, \\ &p^2 H_{21} + d H_{22} + 3 H_{27} - m_1^2 H = A(m_2^2) A(m_3^2)\,. \end{aligned} \tag{11}$$

The arguments of the sunset functions in the second relation are all $(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$. ($L$ only for the finite volume part).

We split the functions in an infinite volume part, $\tilde{H}_i$, and a finite volume correction, $\tilde{H}_i^V$, with $H_i = \tilde{H}_i + \tilde{H}_i^V$. The infinite volume part has been discussed above. For the finite volume parts we define

$$\begin{aligned} \tilde{H}^V &= \frac{\lambda_0}{16\pi^2}\left(A^V(m_1^2) + A^V(m_2^2) + A^V(m_3^2)\right) \\ &\quad + \frac{1}{16\pi^2}\left(A^{V\epsilon}(m_1^2) + A^{V\epsilon}(m_2^2) + A^{V\epsilon}(m_3^2)\right) \\ &\quad + H^V\,, \\ \tilde{H}_1^V &= \frac{\lambda_0}{16\pi^2}\frac{1}{2}\left(A^V(m_2^2) + A^V(m_3^2)\right) \\ &\quad + \frac{1}{16\pi^2}\frac{1}{2}\left(A^{V\epsilon}(m_2^2) + A^{V\epsilon}(m_3^2)\right) + H_1^V\,, \\ \tilde{H}_{21}^V &= \frac{\lambda_0}{16\pi^2}\frac{1}{3}\left(A^V(m_2^2) + A^V(m_3^2)\right) \\ &\quad + \frac{1}{16\pi^2}\frac{1}{3}\left(A^{V\epsilon}(m_2^2) + A^{V\epsilon}(m_3^2)\right) + H_{21}^V\,, \\ \tilde{H}_{27}^V &= \frac{\lambda_0}{16\pi^2}\left(A_{23}^V(m_1^2) + \frac{1}{3}A_{23}(m_2^2) + \frac{1}{3}A_{23}^V(m_3^2)\right) \\ &\quad + \frac{1}{16\pi^2}\left(A_{23}^{V\epsilon}(m_1^2) + \frac{1}{3}A_{23}^{V\epsilon}(m_2^2) + \frac{1}{3}A_{23}^{V\epsilon}(m_3^2)\right) \\ &\quad + H_{27}^V\,. \end{aligned} \tag{12}$$

The finite parts are defined differently from the infinite volume case in [16]. The parts with $A^{V\epsilon}$ are removed here as well.

[5] In the cms frame $t_{\mu\nu} = g_{\mu\nu} - p_\mu p_\nu / p^2$ but the given separation appears naturally in the calculation [19]. It also avoids singularities in the limit $p \to 0$.

The functions H_i^V can be computed with the methods of [19]. They correspond to adding the parts labeled with G and H in Sect. 4.3 and the part of Sect. 4.4 in [19].

They are implemented as functions `hhVt,hh1Vt, hh21Vt,hh22Vt,hh27Vt` with arguments `m1sq,m2sq, m3sq,psq,L,mu2`. The derivatives w.r.t. p^2 exist as `hhdVt,hh1dVt,hh21dVt,hh22dVt,hh27dVt`. These are the functions using the theta function method. Those using the Bessel function method are implemented with a b instead of t as last letter in the name. The arguments are the same.

For all cases discussed we have done checks that both methods, via Bessel or (generalized) Jacobi theta functions, give the same results. In addition the derivatives w.r.t. p^2 for all the integrals are compared with taking a numerical derivative.

Note that the sunset functions at finite volume call the tadpole integrals evaluated with the same method. Do not forget to set precision for those as well. The precision for the sunset integrals can be set with the functions `setprecisionfinitevolumesunsett(racc, rsacc,printout)` and `setprecisionfinite volumesunsetb(maxsum1,mxsum2,racc,rsacc, printout)`. The bool variable `printout` defaults to `true` and sets whether the setting is printed. The double values `sunsetracc` and `sunsetrsacc` set the accuracies of the numerical integration needed when one or two loop-momenta "feel" the finite volume. Default values are `1e-5` and `1e-4` respectively. The integers `maxsum1` and `maxsum2` give how far the sum over Bessel functions is used for the same two cases. The first is maximum 400, the second maximum 40 in the present implementation. In the latter case a triple sum is needed, hence the much lower upper bound. For most applications it makes sense to have a higher precision for the case with one loop momentum quantized, i.e. `racc` smaller than `rsacc`.

7 Masses, decay constants and vacuum-expectation-values

7.1 Masses

The masses of the pion, kaon and eta at two-loops in three flavour ChPT were calculated in [16]. The pion and eta mass were done earlier with a different subtraction scheme and a different way to perform the sunset integrals in [22].

The expressions for the physical masses for $a = \pi, K, \eta$ are given by

$$m_{a\,\mathrm{phys}}^2 = m_{a\,0}^2 + m_a^{2(4)} + m_a^{2(6)}\,. \tag{13}$$

The superscripts indicate the order of the diagrams in p that each contribution comes from. The lowest order masses are

given in (4). The expressions can be found in [16]. In addition the contributions themselves are split in the parts depending on the NLO LECs L_i^r, on the NNLO LECs C_i^r and the remainder as

$$m_a^{2(4)} = m_{a\,L}^{2(4)} + m_{a\,R}^{2(4)}\,, \qquad m_a^{2(6)} = m_{a\,L}^{2(6)} + m_{a\,C}^{2(6)} + m_{a\,R}^{2(6)}\,. \tag{14}$$

All the parts in (14) are implemented as the functions `mpi4(physmass,Li)`, `mpi4L(physmass,Li)`, `mpi4R(physmass)`, `mpi6(physmass,Li,Ci)`, `mpi6L(physmass,Li)`, `mpi6C(physmass,Ci)` and `mpi6R(physmass)`. The equivalent functions also exist for the kaon, with `pi` to `k`, and eta, with `pi` to `eta`.

The functions are defined in `massesdecayvev.h` and implemented in `massesdecayvev.cc`.

7.2 Decay constants

The decay constants of the pion, kaon and eta at two-loops in three flavour ChPT were calculated in [16]. The pion and eta decay constants were done earlier with a different subtraction scheme and a different way to perform the sunset integrals in [22].

The expressions for the decay constants for $a = \pi, K, \eta$ are given by

$$F_{a\,\mathrm{phys}} = F_0\left(1 + F_a^{(4)} + F_a^{(6)}\right)\,. \tag{15}$$

The superscripts indicate the order of the diagrams in p that each contribution comes from. F_0 denotes the decay constant in the three-flavour chiral limit. The expressions were originally derived in [16], but note the description in the erratum of [15]. The expressions corrected for the error can be found in the website [23]. In addition the contributions themselves are split in the parts depending on the NLO LECs L_i^r, on the NNLO LECs C_i^r and the remainder as

$$F_a^{(4)} = F_{a\,L}^{(4)} + F_{a\,R}^{(4)}\,, \qquad F_a^{(6)} = F_{a\,L}^{(6)} + F_{a\,C}^{(6)} + F_{a\,R}^{(6)}\,. \tag{16}$$

All the parts in (16) are implemented as the functions `fpi4(physmass,Li)`, `fpi4L(physmass,Li)`,`fpi4R(physmass)`, `fpi6(physmass,Li,Ci)`, `fpi6L(physmass,Li)`, `fpi6C(physmass,Ci)` and `fpi6R(physmass)`. The equivalent functions also exist for the kaon, with `pi` to `k`, and eta, with `pi` to `eta`. For the η the decay constant has been defined with the octet axial-vector current.

The functions are defined in `massesdecayvev.h` and implemented in `massesdecayvev.cc`.

7.3 `getfpimeta`

A problem that occurs in trying to compare to lattice QCD is that the present routines are written in terms of the physical pion decay constant and masses. In particular, the eta mass is treated as physical. One thus needs a consistent eta mass and pion decay constant when varying the input pion and kaon mass. This assumes we have fitted the LECs L_i^r and C_i^r with a known set of $m_\pi, m_K, m_\eta, F_\pi$.

The functions `getfpimeta6(mpiin,mkin,massin,Li,Ci)` and `getfpimeta4(mpiin,mkin,massin,Li)` return a `physmass` with a consistent set of F_π and m_η for input values of the pion and kaon mass. The other input is the `physmass` `massin`, the `Li` and `Ci` that are used as input. The formulas used are (14) and (16) up to order p^6 and p^4 respectively. The solution is obtained by iteration and stops when six digits of precision are reached. This method was used in [20] to obtain the consistent set of masses and decay constants used there.

7.4 Vacuum-expectation-values

The corrections to the vacuum expectation values (vevs) $\langle 0|\overline{q}q|0\rangle$ for up, down and strange quarks in the isospin limit were calculated at two-loops in three flavour ChPT in [15]. The expression for the up and down quark vev are identical since we are in the isospin limit.

We write the expressions in a form analoguous to the decay constant treatment:

$$\langle 0|\overline{q}q|0\rangle_{a\,\mathrm{phys}} = -F_0^2 B_0\left(1 + \langle 0|\overline{q}q|0\rangle_a^{(4)} + \langle 0|\overline{q}q|0\rangle_a^{(6)}\right)\,. \tag{17}$$

The superscripts indicate the order of the diagrams in p that each contribution comes from. The lowest order values are $-F_0^2 B_0$.

Note that the vevs are not directly measurable quantities. They depend on exactly the way the scalar densities are defined in QCD. ChPT can be used for them when a massin-dependent, chiral symmetry respecting subtraction scheme is used. $\overline{MS}$ in QCD satisfies this, but there are other possibilities. Even within a scheme, B_0 and the quark masses depend on the QCD subtraction scale μ_{QCD} in such a way that $B_0 m_q$ is independent of it. The higher order corrections in this case also depend on the LECs for fully local counterterms, H_1^r, H_2^r at order p^4 and $C_{91}^r, \ldots, C_{94}^r$ at p^6. When the scalar density is fully defined, measuring these quantities in e.g. lattice QCD and comparing with the ChPT expressions is a well defined procedure.

The contributions at the different orders themselves are split in the parts depending on the NLO LECs L_i^r, on the NNLO LECs C_i^r and the remainder as

$$\langle 0|\overline{q}q|0\rangle_a^{(4)} = \langle 0|\overline{q}q|0\rangle_{a\,L}^{(4)} + \langle 0|\overline{q}q|0\rangle_{a\,R}^{(4)},$$
$$\langle 0|\overline{q}q|0\rangle_a^{(6)} = \langle 0|\overline{q}q|0\rangle_{a\,L}^{(6)} + \langle 0|\overline{q}q|0\rangle_{a\,C}^{(6)} + \langle 0|\overline{q}q|0\rangle_{a\,R}^{(6)}. \tag{18}$$

All the parts in (18) are implemented as the functions `qqup4(physmass,Li)`, `qqup4L(physmass,Li)`, `qqup4R(physmass)`,`qqup6(physmass,Li,Ci)`,`qqup6L(physmass,Li)`, `qqup6C(physmass,Ci)` and `qqup6R(physmass)`. The equivalent functions also exist for the strange quark case, with `up` changed to `strange`.

The functions are defined in `massesdecayvev.h` and implemented in `massesdecayvev.cc`.

8 Masses and decay constants at finite volume

The expressions treated in this section have been derived in [20]. A general remark is that care should be taken to set the precision in the loop integrals sufficiently high. For the one-loop integrals setting it very high is usually no problem. For the sunset integrals the evaluation can become very slow. It is strongly recommended to play around with the settings and compare the outputs for the two ways to evaluate the integral. The theta and Bessel function evaluation approach the correct answer differently. For most cases it is possible to have `rsacc` set smaller than `racc`.

For many applications it is useful to calculate the very time consuming parts, those labeled `6RV`, once and store them. They only depend nontrivially on the masses and size of the finite volume. The decay constant dependence is very simple and there is dependence on the NLO LECs L_i^r.

The results presented in this section are with periodic boundary conditions and an infinite extension in the time direction. They are also restricted to the case where the particle is at rest, i.e. $\mathbf{p} = 0$.

8.1 Masses at finite volume

The finite volume corrections to the masses squared[6] are defined as the difference of the mass squared in finite volume and in infinite volume:

$$\Delta^V m_a^2 = m_a^{2V} - m_a^{2\,V=\infty} = m_a^{2V(4)} + m_a^{2V(6)}\,.$$
$$m_a^{2V(6)} = m_{a\,L}^{2V(6)} + m_{a\,R}^{2V(6)}\,. \tag{19}$$

These definitions are for $a = \pi, K, \eta$. These functions are available as `mpi4Vt`, `mpi6Vt`, `mpi6LVt` `mpi6RVt` respectively for the pion. `mpi4Vt` and `mpi6RVt` have as arguments a physmass and the length L. The other two have as arguments `physmass,Li, double L`. The final letter "t" indicates the evaluation using theta functions. With a "b" instead they use evaluation via Bessel functions.

The equivalent functions for the kaon (pi to k) and eta (pi to eta) are also available. All these are defined in `massdecayvevV.h` and implemented in `massdecayvevV.h`.

8.2 Decay constants at finite volume

The finite volume corrections to the decay constants are defined as the difference of the mass squared in finite volume and in infinite volume:

$$\Delta^V F_a = F_a^V - F_a^{V=\infty} = F_a^{V(4)} + F_a^{V(6)}\,.$$
$$F_a^{V(6)} = F_{a\,L}^{V(6)} + F_{a\,R}^{V(6)}\,. \tag{20}$$

These definitions are for $a = \pi, K, \eta$. Note that the correction is defined to the value of the decay constant, not divided by the lowest order decay constant as in (15). The functions are available as `fpi4Vt`, `fpi6Vt`, `fpi6LVt` `fpi6RVt` respectively for the pion. `fpi4Vt` and `fpi6RVt` have as arguments a physmass and the length L. The other two have as arguments `physmass,Li, double L`. The final letter "t" indicates the evaluation using theta functions. With a "b" instead they use evaluation via Bessel functions.

The equivalent functions for the kaon (pi to k) and eta (pi to eta) are also available. All these are defined in `massdecayvevV.h` and implemented in `massdecayvevV.h`.

9 Various comments

9.1 Error handling

Error handling has been dealt with in a very simple manner. Most functions print out a message to standard output if something doesn't seem right. In particular, since the subtraction scale is present in several inputs, many functions check if these are the same and print out a message if not.

Errors due to a zero in a denominator are not caught and might lead to a crash.

9.2 Warnings

The definition of higher orders in ChPT is not unique. In this program collection, we have consistently chosen to rewrite all results in the physical masses and the physical pion decay constant, but note that even this is not fully unique. While there is usually a standard choice for the lowest order expression, at one-loop order this is often not the case since the

[6] Note that in other papers the corrections to the mass itself are sometimes quoted.

Gell-Mann–Okubo relation can be used to rewrite the dependence on the η mass.

The files contain many internal functions as well as some extensions which are not described in this manuscript. These might change in future releases and have in general not been as fully tested as the described ones. Use at your own risk.

9.3 Possible extensions

Two-loop results are known for many more ChPT quantities also including isospin violation as well as for two-flavour ChPT and the partially quenched case. In addition at the one-loop level very many extensions exist for inclusion of the internal electro-magnetic interaction, finite volume effects, twisting and various extensions that include finite a effects in lattice gauge theory.

Another extension is how the higher orders are actually defined. In this program collection so far we have consistently chosen to rewrite all results in the physical masses and the physical pion decay constant. Implementations of other choices of higher orders, in particular in terms of lowest-order quantities are planned.

A last extension worth mentioning is the inclusion of the existing two-flavour ChPT results.

10 Conclusions

This paper describes a library of useful numerical programs in C++ for ChPT at upto two-loop order. Care has been taken to be independent of other libraries. In particular a number of numerical routines has been reimplemented in the numerical algorithm part of the library, `libjbnumlib.a`. The more ChPT direct functions like the loop integrals, a number of data structures to deal with LECs and the result for the masses, decay constants and decay constants are put in `libchiron.a`. Finite volume results are included for the masses and decay constants.

A simple Makefile as well as a large number of testing/example programs are included.

Acknowledgments This work is supported, in part, by the European Community SP4-Capacities "Study of Strongly Interacting Matter" (HadronPhysics3, Grant Agreement number 283286) and the Swedish Research Council Grants 621-2011-5080 and 621-2013-4287. I also thank all my collaborators in the work for which my version of the program made it into this collection and especially Ilaria Jemos who has tested many of the earlier versions in the course of [14].

References

1. S. Weinberg, Physica A **96**, 327 (1979)
2. J. Gasser, H. Leutwyler, Ann. Phys. **158**, 142 (1984)
3. J. Gasser, H. Leutwyler, Nucl. Phys. B **250**, 465 (1985)
4. J. Bijnens, Prog. Part. Nucl. Phys. **58**, 521 (2007). arXiv:hep-ph/0604043
5. http://en.wikipedia.org/wiki/Chiron
6. http://www.gnu.org/licenses/gpl-2.0.html
7. http://www.thep.lu.se/~bijnens/chiron/
8. J. Bijnens, G. Ecker, Ann. Rev. Nucl. Part. Sci. **64**, 149–174 (2014). doi:10.1146/annurev-nucl-102313-025528. arXiv:1405.6488 [hep-ph]
9. http://cernlib.web.cern.ch
10. G. 't Hooft, M.J.G. Veltman, Nucl. Phys. B **153**, 365–401 (1979)
11. A. van Doren, L. de Ridder, J. Comput. Appl. Math. **2**, 207–217 (1976)
12. J. Bijnens, G. Colangelo, G. Ecker, JHEP **9902**, 020 (1999). arXiv:9902437 [hep-ph]
13. J. Bijnens, G. Colangelo, G. Ecker, Ann. Phys. **280**, 100 (2000). arXiv:9907333 [hep-ph]
14. J. Bijnens, I. Jemos, Nucl. Phys. B **854**, 631 (2012). arXiv:1103.5945 [hep-ph]
15. G. Amorós, J. Bijnens, P. Talavera, Nucl. Phys. B **585**, 293 (2000). [Erratum-ibid. B 598 (2001) 665] arXiv:hep-ph/0003258
16. G. Amorós, J. Bijnens, P. Talavera, Nucl. Phys. B **568**, 319 (2000). arXiv:9907264 [hep-ph]
17. J. Bijnens, P. Talavera, JHEP **0203**, 046 (2002). arXiv:0203049 [hep-ph]
18. G. Passarino, M.J.G. Veltman, Nucl. Phys. B **160**, 151 (1979)
19. J. Bijnens, E. Boström, T.A. Lähde, JHEP **1401**, 019 (2014). arXiv:1311.3531 [hep-lat]
20. J. Bijnens, T. Rössler, JHEP **1501**, 034 (2015). doi:10.1007/JHEP01(2015)034. arXiv:1411.6384 [hep-lat]
21. J. Bijnens, K. Ghorbani, Phys. Lett. B **636**, 51 (2006). arXiv:hep-lat/0602019
22. E. Golowich, J. Kambor, Phys. Rev. D **58**, 036004 (1998). arXiv:hep-ph/9710214
23. http://www.thep.lu.se/~bijnens/chpt/