CHIRON v0.55alpha Manual and User Guide

Johan Bijnens

Department of Astronomy and Theoretical Physics, Lund University
Sölvegatan 14A, SE 22362 Lund, Sweden

Manual version of March 12, 2021

Abstract

This manual and user guide describes the classes and functions contained in the ChPT program collection CHIRONv0.55alpha which includes the numerical library `jbnumlib` and the ChPT routine library `chiron`.

Contents

1 Introduction

This is the manual and user guide for the Chiral Perturbation Theory package CHIRON v0.55alpha. It also defines the functions included in a more extended fashion as compared to the published short description [1]. There is obviously a large overlap with that publication. The numerical routines are described in Sect. 4. The remaining sections are devoted to the `chiron` library.
This manual is released under the creative commons license CC-BY 4.0 [2] as reproduced in App. B except for the parts in App. A and App. B which have their own licenses. The software itself is released under the GNU General Public License (GPL) version 2 or later, which is reproduced in App. A.
Kheiron, $X\varepsilon\iota\rho\omega\nu$, or Chiron, was the wisest and eldest of the Centaurs, half-horse men of Greek mythology. His name comes from the Greek word for hand (Kheir) which is also the origin of the word chiral which is why his name was deemed appropriate for this package [3].

2 Guidelines

2.1 Main comments

Most of these routines were produced during and after scientific research. They are licensed under the GPL v2 or later, see App. A, [4] or the file `COPYING` in the main directory, so you have very strong rights in using and modifying them. However, please respect the guidelines as described in the file `GUIDELINES` in the main directory. A summary of these is

- Citations are important in the academic world so when using these please both cite the relevant CHIRON publication [1] and the papers where the work itself was done as quoted in the different chapters.
- Suspected bugs, proposed fixes and suggestions should preferably be communicated to the author(s) so they can be added in future releases.
- If you distribute modified versions, please indicate clearly the modifications in the source and at the point of distributions. However, the preferred way to introduce changes is via future releases.
- To make published results reproducible, the exact versions of the code that were used should be kept. This includes the values of all parameters used including the precisions.

2.2 Some caution for use

These routines have been used and tested in a ChPT environment using units in powers of GeV. Typical accuracies are set by default to relevant and obtainable values for that case. In addition, there are often special cases where the routines might not work, often due to 0/0 or large cancelations.
Similar comments apply to the special functions included. They are sufficiently accurate for the purposes they were used for originally and usually return values with a precision close to double precision but this is not guaranteed.
In some cases, the large formulas have inherently large cancelations. This might lead to degrading of precision in unexpected places. Use common (scientific) sense to judge the quality of the results.
Finally, there are a number of internal functions and extensions already present in the source code but not yet documented in this manual. These might change and have not been tested as well as the documented ones. In particular interfaces etc. might change.

3 Files, installation and testroutines

The package can be downloaded from [5]. There are ready to install libaries there for some cases, but in general it is better to compile it for your own system. C++ can have a large overhead in calling classes and functions compared to FORTRAN. Therefore always compile the library with optimization. The interfaces are as much as possible defined with the keyword `const` to allow the compiler to optimize more efficiently.

3.1 Files

The gzipped tarred file (`chiron.vvvv.tar.gz`) will produce a directory `chiron.vvvv` with a number of subdirectories. `vvvv` is version information. The created directory is called the main directory in the remainder.
The main directory contains the files `COPYING`, `INSTRUCTIONS`, `GUIDELINES` and a `Makefile`.
The subdirectory `doc` contains the documentation. The latest published article about CHIRON, this manual (`manual.tex`), a list of files (`filelist.txt`) and a summary of things added since earlier versions (`Changelog.vvv.to.www.txt`).
The subdirectory `lib` will after compiling contain the compiled libraries `libjbnumlib.a` and `libchiron.a`.
The subdirectory `include` contains all the needed header files. The subdirectory `src` contains the source files. `test` contains the testing and example programs. `testoutputs` contains the output the testprograms should produce.
Typically for each subject `xxx` there are files `xxx.h`, `xxx.cc`, `testxxx.cc` and `testxxx.dat` in the respective directories.
There are a few extra files around as well. These typically contain inputs needed or large sets of constants.

3.2 Installation

The main steps are to run make in the main directory. This should produce the files `libjbnumlib.a` and `libchiron.a` and also copy them to the `lib` subdirectory. You might have to change the variables `CXX`, `CFLAGS` and `CFLAGTESTS`. `CXX` should specify the C++ compiler and the options to be used for everything. `CFLAGS` can be used to specify additional options in compiling the libraries and `CFLAGTESTS` to specify additional options for the testing programs.
"make clean" can be used to remove many of the files created during compiling.
The actual installation is by putting the contents of the `include` directory somewhere in the include path of your compiler and the two files `libjbnumlib.a` and `libchiron.a` somewhere in the library path. For many C++ compilers the paths are given in the environment variables `CPLUS_INCLUDE_PATH` and `LIBRARY_PATH` respectively.

3.3 testroutines

For every file `xxx.h` and `xxx.cc` included for `chiron` there is a testing/example code `testxxx.cc` in the subdirectory `test`. These can be compiled using "make testxxx" in the main directory. Executing the resulting file `a.out` should then produce output identical (up to the precision specified and possible randomly generated cases) to the file `testxxx.dat` in the subdirectory `testoutputs`.

4 `jbnumlib`

4.1 Complex numbers

Complex numbers are defined via the standard C++ library and an abbreviation provided as

```
typedef std::complex<double> dcomplex;
```

All variables declared complex will be of the this type and referred to as `dcomplex` in the remainder.

4.2 Special functions

4.2.1 Polylogarithms

4.2.1.1 `jbdli2p`

```
dcomplex jbdli2p(const dcomplex x)
```

Returns the complex dilogarithm or Spence function defined by

$$\mathrm{Li}_2(x) = -\int_0^1 dt \frac{\log(1-xt)}{t}\,, \tag{1}$$

where it converges and analytic continuation. Cut defined on the positive real axis from 1 to ∞. Uses the properties of the dilogarithm to transform the argument and then the Bernouilly series as described in [6].
Defined in `jbnumlib.h` and implemented in `jbdli2.cc`. This was `jbdli2` in versions of `jbnumlib` up to 0.54.

4.2.1.2 `jbdli2`

```
dcomplex jbdli2(const dcomplex x)
```

Returns the complex dilogarithm or Spence function defined by

$$\mathrm{Li}_2(x) = -\int_0^1 dt \frac{\log(1-xt)}{t}\,, \tag{2}$$

where it converges and analytic continuation. Cut defined on the positive real axis from 1 to ∞. Uses the properties of the dilogarithm to transform the argument and then the Bernouilly series around zero or 1 as described in [8]. Somewhat faster than `jbdli2p` in some cases.
Defined in `jbnumlib.h` and implemented in `jbdlin.cc`.

4.2.1.3 `jbdli3`

```
dcomplex jbdli3(const dcomplex x)
```

Returns the complex trilogarithm function defined by

$$\mathrm{Li}_3(x) = \int_0^1 dt \frac{\mathrm{Li}_2(xt)}{t} \,, \tag{3}$$

where it converges and analytic continuation. Cut defined on the positive real axis from 1 to ∞. Uses the properties of the trilogarithm to transform the argument and then the Bernouilly series around zero or 1 as described in [8].
Defined in `jbnumlib.h` and implemented in `jbdlin.cc`.

4.2.1.4 `jbdli4`

`dcomplex jbdli4(const dcomplex x)`
Returns the complex polylogarithm function defined by

$$\mathrm{Li}_4(x) = \int_0^1 dt \frac{\mathrm{Li}_3(xt)}{t} \,, \tag{4}$$

where it converges and analytic continuation. Cut defined on the positive real axis from 1 to ∞. Uses the properties of the trilogarithm to transform the argument and then the Bernouilly series around zero or 1 as described in [8].
Defined in `jbnumlib.h` and implemented in `jbdlin.cc`.

4.2.2 Bessel functions

4.2.2.1 `jbdbesi0`

`double jbdbesi0(const double x)`
Returns the modified Bessel function I_0 for real values of the argument. A simple port to C++ of CERNLIB[7] routine DBESI0.
Defined in `jbnumlib.h`, implemented in `jbdbesik.cc`.

4.2.2.2 `jbdbesi1`

`double jbdbesi1(const double x)`
Returns the modified Bessel function I_1 for real values of the argument. A simple port to C++ of CERNLIB[7] routine DBESI1.
Defined in `jbnumlib.h`, implemented in `jbdbesik.cc`.

4.2.2.3 `jbdbesk0`

`double jbdbesk0(const double x)`
Returns the modified Bessel function K_0 for real values of the argument. A simple port to C++ of CERNLIB[7] routine DBESK0.
Defined in `jbnumlib.h`, implemented in `jbdbesik.cc`.

4.2.2.4 `jbdbesk1`

`double jbdbesk1(const double x)`
Returns the modified Bessel function K_1 for real values of the argument. A simple port to C++ of CERNLIB[7] routine DBESK1.
Defined in `jbnumlib.h`, implemented in `jbdbesik.cc`.

4.2.2.5 `jbdbesk2`

`double jbdbesk2(const double x)`
Returns the modified Bessel function K_2 for real values of the argument. Uses the recursion relations for Bessel functions and `jbdbesk0` and `jbdbesk1`.
Defined in `jbnumlib.h`, implemented in `jbdbesik.cc`.

4.2.2.6 `jbdbesk3`

`double jbdbesk3(const double x)`
Returns the modified Bessel function K_3 for real values of the argument. Uses the recursion relations for Bessel functions and `jbdbesk0` and `jbdbesk1`.
Defined in `jbnumlib.h`, implemented in `jbdbesik.cc`.

4.2.2.7 `jbdbesk4`

`double jbdbesk4(const double x)`
Returns the modified Bessel function K_4 for real values of the argument. Uses the recursion relations for Bessel functions and `jbdbesk0` and `jbdbesk1`.
Defined in `jbnumlib.h`, implemented in `jbdbesik.cc`.

4.2.3 Theta and related functions

4.2.3.1 `jbdtheta30`

`double jbdtheta30(const double q)`
Returns the value of the function

$$\theta_{30}(q) = 1 + 2\sum_{n=1,\infty} q^{(n^2)} = \sum_{n=-\infty,\infty} q^{(n^2)}\,. \tag{5}$$

This function is related to the third Jacobi theta function. For small q the summation in (5) is used directly. For larger q the identity

$$\theta_{30}(q) = \sqrt{\frac{\lambda}{\pi}}\theta_{30}\left(e^{-\lambda}\right) \tag{6}$$

with $\lambda = \pi^2/|\log(q)|$ is used instead. This is related to the modular invariance for the higher dimensional case. Precision can be judged by comparing the two series to each other. Same idea as used in the CERNLIB[7] routine `DTHETA`.
Defined in `jbnumlib.h`, implemented in `jbdtheta30.cc`.

4.2.3.2 `jbdtheta30m1`

`double jbdtheta30m1(const double q)`
Returns the value of the function

$$\theta_{30}(q) - 1 = 2 \sum_{n=1,\infty} q^{(n^2)} = \sum_{n\in\mathbb{Z},n\neq 0} q^{(n^2)}\,. \qquad (7)$$

Implementation as for `jbdtheta30` but without the 1. Especially for small q often needed to keep accuracy in the finite volume applications in ChPT.
Defined in `jbnumlib.h`, implemented in `jbdtheta30m1.cc`.

4.2.3.3 `jbdtheta32`

`double jbdtheta32(const double q)`
Returns the value of the function

$$\theta_{32}(q) = 2 \sum_{n=1,\infty} q^{(n^2)} = \sum_{n=-\infty,\infty} n^2 q^{(n^2)} = q\frac{d}{dq}\theta_{30}(q)\,. \qquad (8)$$

For small q the summation in (8) is used directly. For larger q the derivative of the right-hand-side of the identity (6) is used.
Defined in `jbnumlib.h`, implemented in `jbdtheta32.cc`.

4.2.3.4 `jbdtheta34`

`double jbdtheta34(const double q)`
Returns the value of the function

$$\theta_{34}(q) = \sum_{n=1,\infty} n^4 q^{(n^2)} = \sum_{n=-\infty,\infty} n^4 q^{(n^2)} = \left(q\frac{d}{dq}\right)^2 \theta_{30}(q)\,. \qquad (9)$$

For small q the summation in (9) is used directly. For larger q the appropriate derivative of the right-hand-side of the identity (6) is used.
Defined in `jbnumlib.h`, implemented in `jbdtheta34.cc`.

4.2.3.5 `jbdtheta3`

`double jbdtheta3(const double u, const double q)`
Returns the value of the function

$$\theta_3(u,q) = 1 + 2 \sum_{n=1,\infty} q^{(n^2)} \cos(2\pi n u) = \sum_{n=-\infty,\infty} q^{(n^2)} e^{i2\pi n u}\,. \qquad (10)$$

For small q the summation in (10) is used directly. For larger q the expansion after using the relation

$$\theta_3(u,q) = \sqrt{\pi/|\log(q)|}\exp(-\pi^2 u^2/|\log(q)|)\theta_3(-iu\pi/|\log(q)|, \exp(-\pi^2/|\log(q)|)) \qquad (11)$$

is used.
Defined in `jbnumlib.h`, implemented in `jbdtheta3.cc`.

4.2.3.6 `jbderivutheta3`

`double jbderivutheta3(const double u, const double q)`
Returns the value of the function

$$\frac{\partial}{\partial u}\theta_3(u,q) = -4\pi \sum_{n=1,\infty} q^{(n^2)}\sin(2\pi n u) = i2\pi \sum_{n=-\infty,\infty} n q^{(n^2)} e^{i2\pi n u}\,. \tag{12}$$

For small q the summation in (12) is used directly. For larger q the appropriate derivative of the relation (11) is used.
Defined in `jbnumlib.h`, implemented in `jbderivutheta3.cc`.

4.2.3.7 `jbderiv2utheta3`

`double jbderiv2utheta3(const double u, const double q)`
Returns the value of the function

$$\frac{\partial^2}{\partial u^2}\theta_3(u,q) = -8\pi^2 \sum_{n=1,\infty} q^{(n^2)}\cos(2\pi n u) = -4\pi^2 \sum_{n=-\infty,\infty} n^2 q^{(n^2)} e^{i2\pi n u}\,. \tag{13}$$

For small q the summation in (13) is used directly. For larger q the appropriate derivative of the relation (11) is used.
Defined in `jbnumlib.h`, implemented in `jbderiv2utheta3.cc`.

4.2.3.8 `jbderiv3utheta3`

`double jbderiv3utheta3(const double u, const double q)`
Returns the value of the function

$$\frac{\partial^3}{\partial u^3}\theta_3(u,q) = 16\pi^2 \sum_{n=1,\infty} q^{(n^2)}\sin(2\pi n u) = -8i\pi^3 \sum_{n=-\infty,\infty} n^3 q^{(n^2)} e^{i2\pi n u}\,. \tag{14}$$

For small q the summation in (14) is used directly. For larger q the appropriate derivative of the relation (11) is used.
Defined in `jbnumlib.h`, implemented in `jbderiv2utheta3.cc`.

4.2.3.9 `jbdtheta2d0`

`double jbdtheta2d0(const double a, const double b, const double c)`
Returns the value of the function

$$\theta_0^{(2)}(a,b,c) = \sum_{n_1,n_2=-\infty,\infty} e^{-an_1^2-bn_2^2-c(n_1-n_2)^2}\,. \tag{15}$$

There are many higher-dimensional generalizations of the Jacobi theta functions. The modular invariance properties of these are discussed in App. B of [9] and are used in the evaluation to speed up the calculation. It should be noted that $\theta_0^{(2)}(a,b,c)$ is fully symmetric in a,b,c.
Defined in `jbnumlib.h` and implemented in `jbdtheta2d0.cc`.

4.2.3.10 `jbdtheta2d0m1`

`double jbdtheta2d0m1(const double a, const double b, const double c)`
Returns the value of the function

$$\theta_0^{(2)}(a,b,c) - 1 = \sum_{n_1,n_2=-\infty,\infty} e^{-an_1^2-bn_2^2-c(n_1-n_2)^2} - 1 = \sum_{\substack{n_1,n_2\in\mathbb{Z}\\(n_1,n_2)\neq(0,0)}} e^{-an_1^2-bn_2^2-c(n_1-n_2)^2} - 1\,. \tag{16}$$

Method as in `jbdtheta2d0` but the 1 removed, more accurate for small a, b, c as often needed in finite volume ChPT.
Defined in `jbnumlib.h` and implemented in `jbdtheta2d0m1.cc`.

4.2.3.11 `jbdtheta2d02`

`double jbdtheta2d02(const double a, const double b, const double c)`
Returns the value of the function

$$\theta_{02}^{(2)}(a,b,c) = \sum_{n_1,n_2=-\infty,\infty} n_1^2 e^{-an_1^2-bn_2^2-c(n_1-n_2)^2} = -\frac{\partial}{\partial a}\theta_0^{(2)}(a,b,c)\,. \tag{17}$$

It should be noted that $\theta_{02}^{(2)}(a,b,c)$ is symmetric in b, c. Method similar to `jbdtheta2d0`.
Defined in `jbnumlib.h` and implemented in `jbdtheta2d02.cc`.

4.3 Root finders

4.3.1 Zero of a real function of one variable

4.3.1.1 `jbdzerox`

```
double jbdzerox(double (*f)(const double),const double a0,const double b0,
          const double eps,const int maxf=10000, const int mode=1)
```
Returns the position of a zero of the function `f` in the interval [`a0`,`b0`]. Requires that `f(a0)` and `f(b0)` have different sign.
`maxf` gives the maximum number of function eveluations and `eps` is the precision, relative for results larger than one in absolute value else absolute precision. There are two methods available chosen by mode. 0 is algorithm M and 1 is algorithm R of [10]. Method R converges faster, method M is a bit more robust.
Defined in `jbnumlib.h`, implemented in `jbdzerox.cc`. A short example program is `testjbdzerox.cc`.

4.3.2 Roots of a cubic equation

4.3.2.1 `jbdrteq3`

```
double jbdrteq3(const double r,const double s,const double t,
          double x[3], double & d)
```

Finds the roots of the equation with real coefficients

$$x^3 + rx^2 + sx + t = 0\,. \tag{18}$$

If $d \leq 0$ the three real roots are in $x[0], x[1], x[2]$. For degenerate roots $d = 0$. For $d > 0$ the real root is in $x[0]$ and the two complex conjugate roots are $x[1] \pm ix[2]$.
The routine uses the classical method of Tartaglia-Vieta. Sometimes, the solutions are improved by Newton iteration. A simple port to C++ of CERNLIB[7] routine DRTEQ3.
Defined in `jbnumlib.h`, implemented in `jbdrteq3.cc`. A short example program is `testjbdrteq3.cc`.

4.4 Integration routines

4.4.1 One dimension, real

The interface of these routines is identical so they can be simply interchanged. For most problems the speed decreases as `jbdquad15`, `jbdquad21`, `jbdgauss2`, `jbdgauss` but this is somewhat dependent on the function integrated and the precision requested.
The routines do not use the endpoints so an integrable singularity at the endpoint can be done but an integrand transformation that removes the singularity will lead to a much better performance.
An example program that shows the relative speeds is in `testintegralsreal.cc`.

4.4.1.1 `jbdgauss`

`double jbdgauss(f,a,b,eps)`
`double jbdgauss(fp,a,b,eps,ap)`
Two versions exist:
`f`: `double (*f)(const double x)` The double precision function to be integrated over.
`fp`: `double (*fp)(const double x, void*)` The double precision function to be integrated over, the pointer allows for extra parameters to be transferred through the integration routine.
`a,b,eps`: `const double`
`a`: Lower limit of integration.
`b`: Upper limit of integration.
`eps`: Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
`ap`: `void*`
Subroutine translated from the CERNLIB[7] routine `DGAUSS`. Uses 8 and 16 point Gaussian rules with the 16 point for the estimate and the difference for the error estimate. Adaptive with a subdivision strategy.
Defined in `jbnumlib.h`, implemented in `jbdgauss.cc`.

4.4.1.2 `jbdgauss2`

`double jbdgauss2(f,a,b,eps)`
`double jbdgauss2(fp,a,b,eps,ap)`
Two versions exist:
`f`: `double (*f)(const double x)` The double precision function to be integrated over.
`fp`: `double (*fp)(const double x, void*)` The double precision function to be integrated over, the pointer allows for extra parameters to be transferred through the integration routine.
`a,b,eps`: `const double`
`a`: Lower limit of integration.
`b`: Upper limit of integration.
`eps`: Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
`ap`: `void*`
Uses 8 and 16 point Gaussian rules with the 16 point for the estimate and the difference for the error estimate. Adaptive with a subdivision strategy. Very similar to `jbdgauss` but the subdivision strategy is more appropriate for high precision.
Defined in `jbnumlib.h`, implemented in `jbdgauss2.cc`.

4.4.1.3 `jbdquad15`

`double jbdquad15(f,a,b,eps)`
`double jbdquad15(fp,a,b,eps,ap)`
Two versions exist:
`f`: `double (*f)(const double x)` The double precision function to be integrated over.
`fp`: `double (*fp)(const double x, void*)` The double precision function to be integrated over, the pointer allows for extra parameters to be transferred through the integration routine.
`a,b,eps`: `const double`
`a`: Lower limit of integration.
`b`: Upper limit of integration.
`eps`: Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
`ap`: `void*`
Uses 15 point Gauss-Kronrod rule for the estimate and the difference with the embedded 7 point Gauss rule for the error estimate. Adaptive with a subdivision strategy appropriate for high precision.
Defined in `jbnumlib.h`, implemented in `jbdquad15.cc`.

4.4.1.4 `jbdquad21`

`double jbdquad21(f,a,b,eps)`
`double jbdquad21(fp,a,b,eps,ap)`

Two versions exist:
`f`: `double (*f)(const double x)` The double precision function to be integrated over.
`fp`: `double (*fp)(const double x, void*)` The double precision function to be integrated over, the pointer allows for extra parameters to be transferred through the integration routine.
`a,b,eps`: `const double`
`a`: Lower limit of integration.
`b`: Upper limit of integration.
`eps`: Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
`ap`: `void*`
Uses 21 point Gauss-Kronrod rule for the estimate and the difference with the embedded 10 point Gauss rule for the error estimate. Adaptive with a subdivision strategy appropriate for high precision.
Defined in `jbnumlib.h`, implemented in `jbdquad21.cc`.

4.4.2 One dimension, real with singularity

The interface of these routines is identical so they can be simply interchanged. For most problems the speed decrases as `jbdquad15` or `jbdquad21`, `jbdgauss2`, `jbdgauss` but this is somewhat dependent on the function integrated and the precision requested.
An example program that shows the relative speeds is in `testintegralsrealsingular.cc`.

4.4.2.1 `jbdcauch`

`double jbdcauch(f,a,b,s,eps)`
`f`: `double (*f)(const double x)` The double precision function to be integrated over.
`a,b,s,eps`: `const double`
`a`: Lower limit of integration.
`b`: Upper limit of integration.
`s`: Place of the singularity.
`eps`: Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
Subroutine translated from the CERNLIB[7] routine `DCAUCH`.Uses 8 and 16 point Gaussian rules with the 16 point for the estimate and the difference for the error estimate. Adaptive with a subdivision strategy. Integrates symmetrically around the singularity so it returns the integral in the sense of the principal value prescription. Uses `jbdgauss`.
Defined in `jbnumlib.h`, implemented in `jbdcauch.cc`.

4.4.2.2 `jbdcauch2`

`double jbdcauch2(f,a,b,s,eps)`
`f`: `double (*f)(const double x)` The double precision function to be integrated over.
`a,b,s,eps`: `const double`

`a`: Lower limit of integration.
`b`: Upper limit of integration.
`s`: Place of the singularity.
`eps`: Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
Subroutine translated from the CERNLIB[7] routine `DCAUCH`.Uses 8 and 16 point Gaussian rules with the 16 point for the estimate and the difference for the error estimate. Adaptive with a subdivision strategy more suitable for high precision. Integrates symmetrically around the singularity so it returns the integral in the sense of the principal value prescription. Uses `jbdgauss2`.
Defined in `jbnumlib.h`, implemented in `jbdcauch2.cc`.

4.4.2.3 `jbdsing15`

`double jbdsing15(f,a,b,s,eps)`
`f`: `double (*f)(const double x)` The double precision function to be integrated over.
`a,b,s,eps`: `const double`
`a`: Lower limit of integration.
`b`: Upper limit of integration.
`s`: Place of the singularity.
`eps`: Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
Subroutine similar to `jbdcauch2` but uses a Gauss-Kronrod 15 point rule for the estimate and the difference withe embedded 7 point Gauss rule for the error estimate. Adaptive with a subdivision strategy more suitable for high precision. Integrates symmetrically around the singularity so it returns the integral in the sense of the principal value prescription. Uses `jbdquad15`.
Defined in `jbnumlib.h`, implemented in `jbdsing15.cc`.

4.4.2.4 `jbdsing21`

`double jbdsing21(f,a,b,s,eps)`
`f`: `double (*f)(const double x)` The double precision function to be integrated over.
`a,b,s,eps`: `const double`
`a`: Lower limit of integration.
`b`: Upper limit of integration.
`s`: Place of the singularity.
`eps`: Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
Subroutine similar to `jbdcauch2` but uses a Gauss-Kronrod 21 point rule for the estimate and the difference withe embedded 10 point Gauss rule for the error estimate. Adaptive with a subdivision strategy more suitable for high precision. Integrates symmetrically around the singularity so it returns the integral in the sense of the principal value prescrip-

tion. Uses `jbdquad21`.
Defined in `jbnumlib.h`, implemented in `jbdsing21.cc`.

4.4.3 One dimension, complex

The interface of these routines is identical so they can be simply interchanged. For most problems the speed decrases as `jbwquad15` or `jbwquad21` or `jbwgauss2`, `jbwgauss` but this is somewhat dependent on the function integrated and the precision requested.
An example program that shows the relative speeds is in `testintegralscomplex.cc`.

4.4.3.1 `jbwgauss`

`dcomplex jbwgauss(f,a,b,eps)`
`dcomplex jbwgauss(fp,a,b,eps,ap)`
`f`: `dcomplex (*f)(const dcomplex x)` The complex double precision function to be integrated over.
`fp`: `dcomplex (*fp)(const dcomplex x, void*)` The complex double precision function to be integrated over. The void* allows for extra parameters to be passed to the function.
`a,b`: `const dcomplex`
`a`: Lower endpoint of integration.
`b`: Upper endpoint of integration.
`eps`: `const double` Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
`ap`: `void*`
Subroutine translated from the CERNLIB[7] routine `WGAUSS`. Uses 8 and 16 point Gaussian rules with the 16 point for the estimate and the difference for the error estimate. Adaptive with a subdivision strategy. The integration is the lineintegral over the straight line between `a` and `b`.
Defined in `jbnumlib.h`, implemented in `jbwgauss.cc`.

4.4.3.2 `jbwgauss2`

`dcomplex jbwgauss2(f,a,b,eps) dcomplex jbwgauss2(fp,a,b,eps,ap)`
`f`: `dcomplex (*f)(const dcomplex x)` The complex double precision function to be integrated over.
`fp`: `dcomplex (*fp)(const dcomplex x, void*)` The complex double precision function to be integrated over. The void* allows for extra parameters to be passed to the function.
`a,b`: `const dcomplex`
`a`: Lower endpoint of integration.
`b`: Upper endpoint of integration.
`eps`: `const double` Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
`ap`: `void*`

Subroutine translated from the CERNLIB[7] routine `WGAUSS`. Uses 8 and 16 point Gaussian rules with the 16 point for the estimate and the difference for the error estimate. Adaptive with a subdivision strategy better suited for high precision. The integration is the lineintegral over the straight line between `a` and `b`.
Defined in `jbnumlib.h`, implemented in `jbwgauss2.cc`.

4.4.3.3 `jbwquad15`

`dcomplex jbwquad15(f,a,b,eps)`
`dcomplex jbwquad15(fp,a,b,eps,ap)`
`f`: `dcomplex (*f)(const dcomplex x)` The complex double precision function to be integrated over.
`fp`: `dcomplex (*fp)(const dcomplex x, void*)` The complex double precision function to be integrated over. The void* allows for extra parameters to be passed to the function.
`a,b`: `const dcomplex`
`a`: Lower endpoint of integration.
`b`: Upper endpoint of integration.
`eps`: `const double` Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
`ap`: `void*`
Subroutine similar to `jbwgauss2` but uses a 15 point Gauss-Kronrod rule for the estimate and the difference with the embedded 7 point Gauss rule for the error estimate. Adaptive with a subdivision strategy better suited for high precision. The integration is the lineintegral over the straight line between `a` and `b`.
Defined in `jbnumlib.h`, implemented in `jbwquad15.cc`.

4.4.3.4 `jbwquad21`

`dcomplex jbwquad21(f,a,b,eps)`
`dcomplex jbwquad21(fp,a,b,eps,ap)`
`f`: `dcomplex (*f)(const dcomplex x)` The complex double precision function to be integrated over.
`fp`: `dcomplex (*fp)(const dcomplex x, void*)` The complex double precision function to be integrated over. The void* allows for extra parameters to be passed to the function.
`a,b`: `const dcomplex`
`a`: Lower endpoint of integration.
`b`: Upper endpoint of integration.
`eps`: `const double` Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
`ap`: `void*`
Subroutine similar to `jbwgauss2` but uses a 21 point Gauss-Kronrod rule for the estimate and the difference with the embedded 10 point Gauss rule for the error estimate. Adaptive with a subdivision strategy better suited for high precision. The integration is the

lineintegral over the straight line between `a` and `b`.
Defined in `jbnumlib.h`, implemented in `jbwquad21.cc`.

4.4.4 Two dimensions, real

4.4.4.1 `jbdad2`

`double jbdad2(f,a,b,eps, relerr, ifail)`
`double jbdad2(f,ap,a,b,eps, relerr, ifail)`
`f`: `double (*f)(double x[])` or `double (*f)(double x[],void*)` The double precision function to be integrated over, `x[0]` and `x[1]` contain the values of the two variables to be integrated over. The `void* ap` allows extra parameters to be transferred.
`a`: `double a[]` `a[0]` and `a[1]` are the lower limits of integration.
`b`: `double b[]` `b[0]` and `b[1]` are the upper limits of integration.
`eps`: `const double` Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
`relerr`: `double &` returns the obtained relative precision via a reference.
`ifail`: `int &` returns an integer. Zero indicates success, if not zero the routine did not obtain the requested precision..
The function does a two dimensional integration over a hypercube. The underlying routine is `jbdadmul` which is a simple port to C++ of the CERNLIB[7] routine `DADMUL`. This in turn was based on [11].
Defined in `jbnumlib.h`, implemented in `jbdadmul.cc` and `jbdadmul2.cc`.

4.4.5 Three dimensions, real

4.4.5.1 `jbdad3`

`double jbdad3(f,a,b,eps, relerr, ifail)`
`double jbdad3(f,ap,a,b,eps, relerr, ifail)`
`f`: `double (*f)(double x[])` or `double (*f)(double x[],void*)` The double precision function to be integrated over, `x[0]`, `x[1]` and `x[2]` contain the values of the three variables to be integrated over. The `void* ap` allows extra parameters to be transferred.
`a`: `double a[]` `a[0]`, `a[1]` and `a[2]` are the lower limits of integration.
`b`: `double b[]` `b[0]`, `b[1]` and `b[2]` are the upper limits of integration.
`eps`: `const double` Precision attempted to be reached: relative precision if absolute value of the integral is above 1, otherwise absolute precision.
`relerr`: `double &` returns the obtained relative precision via a reference.
`ifail`: `int &` returns an integer. Zero indicates success, if not zero the routine did not obtain the requested precision..
The function does a three dimensional integration over a hypercube. The underlying routine is `jbdadmul` which is a simple port to C++ of the CERNLIB[7] routine `DADMUL`. This in turn was based on [11].
Defined in `jbnumlib.h`, implemented in `jbdadmul.cc` and `jbdadmul2.cc`.

5 Chiral Perturbation Theory

The classic papers introducing ChPT are [12, 13, 14]. References to lectures and introductions can be found in [15]. A review at two-loop order is [16]. The notation used here correspond to the notation introduced by Gasser and Leutwyler, B, F, l_i^r [13] and B_0, F_0, L_i^r [14] for the two and three flavour case. In general the decay constants are defined with a normalization of $F_\pi \approx 92$ MeV. The coupling constants in the higher order Lagrangians are usually referred to as low-energy constants (LECs). Power counting is the usual dimensional counting with orders referred to as p^n with alternatively p^2 or lowest-order (LO), p^4 or next-to-leading-order (NLO) and p^6 or next-to-next-to-leading order (NNLO).

6 Data structures

This section describes a number of classes to deal with input parameters and LECs. The default value mechanism of C++ is used to give them initial values if not specified. These are visible below as "=value" in the definitions.

6.1 Two flavour ChPT

6.1.1 Class: `physmassnf2`

`physmassnf2(mpiin=0.135,fpiin=0.0922,muin=0.77)`
`mpiin,fpiin,muin:  const double`
Private data: `double mpi,fpi,mu`
Physical quantities: pion, pion-decay constant and subtraction scale μ.
Relevant physical case: two flavour ChPT, isospin limit.

Input member functions:

```
void setmpi(const double mpiin=0.135)
void setfpi(const double fpiin=0.0922)
void setmu(const double muiin=0.77)
```

Output member functions exist in two varieties. Those that return all or a subset of values using references or those that return one value as the function value.

```
void out(double &mpiout, double &fpiout,
      double &muout)
double getmpi(void)
double getfpi(void)
double getmu(void)
```

Operators defined: `<<`, `>>` and `==`.
`<<` and `>>` are defined such that output and input streams work as expected. The input

stream should be exactly in the format provided by the output stream.
== checks for equality within relative precision of 10^{-7}. An error will occur if any of the data members is zero.

Defined in `inputsnf2.h`, implemented in `inputsnf2.cc`, examples of use in `testinputsnf2.cc`.

6.1.2 Class: `lomassnf2`

`lomassnf2(mp0in=0.135, muin=0.77)`
`mp0in,muin:  const double`
`lomassnf2(const quarkmassnf2 mass)`
Private data: `double mp,f,mu`
Physical quantities: lowest order pion mass, lowest order pion-decay constant and subtraction scale μ.
Relevant physical case: two flavour ChPT, isospin limit.

The constructor from a `quarkmassnf2` is provided such that conversions can be used.
Input member functions:
`void setmp(const double mp0in=0.135) void setf(const double f0in=0.09) void setmu(const double muin=0.77)`

Output member functions exist in two varieties. Those that return all or a subset of values using references or those that return one value as the function value.

```
void out(double &mp0out, double &f0out, double &muout)
double getmp(void)
double getf(void)
double getmu(void)
```

Operators defined: <<, >> and ==.
<< and >> are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
== checks for equality within relative precision of 10^{-7}. An error will occur if any of the data members is zero.

Defined in `inputsnf2.h`, implemented in `inputsnf2.cc`, examples of use in `testinputsnf2.cc`.

6.1.3 Class: `quarkmassnf2`

`quarkmassnf2(B0mhatin=0.01, f0in=0.090, muin=0.77)`
`B0mhatin,f0in,muin:  const double`
`quarkmassnf2(const lomassnf2 mass)`
Private data: `double Bmhat,f,mu`
Physical quantities: $B\hat{m}$, lowest order pion-decay constant and subtraction scale μ.

The quantities $B\hat{m}$ is the LEC B [13] multiplied by the average up-down quark mass and strange quark mass respectively. These are independent of the QCD scale. The lowest order pion mass are given by $m_{\pi\,\mathrm{LO}} = \sqrt{2B_0\hat{m}}$.
Relevant physical case: two flavour ChPT, isospin limit.

The constructor from a `lomassnf2` is provided such that conversions can be used.
Input member functions:

```
void setBmhat(const double B0mhatin=0.01)
void setf(const double f0in=0.09)
void setmu(const double muin=0.77)
```

Output member functions exist in two varieties. Those that return all or a subset of values using references or those that return one value as the function value.

```
void out(double &B0mhatout, double &f0out, double &muout)
double getBmhat(void)
double getf(void)
double getmu(void)
```

Operators defined: `<<`, `>>` and `==`.
`<<` and `>>` are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
`==` checks for equality within relative precision of 10^{-7}. An error will occur if any of the data members is zero.

Defined in `inputsnf2.h`, implemented in `inputsnf2.cc`, examples of use in `testinputsnf2.cc`.

6.1.4 NLO LECs: Class `li`

```
li(l1r=0.,l2r=0.,l3r=0.,l4r=0.,l5r=0.,l6r=0.,l7r=0.,h1r=0.,h2r=0.,h3r=0.,
   mu=0.77,Name="Nameless li")
li(liin, mu = 0.77, mpi= 0.13957061)
const double:  l1r,...,l7r,h1r,h2r,h3r,mu,mpi
const string:  Name
const libar:  liin
```

Protected data: `double l1r,l2r,l3r,l4r,l5r,l6r,l7r,h1r,h2r,h3r,mu` and `string name`
Physical quantities the 10 LECs, l_i^r, h_i^r (of which three are so-called contact terms) of two-flavour ChPT as introduced in [13] and the subtraction scale μ.
Relevant physical case: two flavour ChPT

Input member functions:

```
void setli(const int n, const double lin)
void setli(const double lin, const int n)
```

Set the value of the LECs with index n. $n = 8, 9, 10$ correspond to h^r_1, h^r_2, h^r_3.
`setmu(const double muin)`
Sets the scale μ to the value `muin`. This does *not* change the LECs, for that use `changescale`.
`setname(const string namein)` Sets the name of the set of LECs.

Output member functions:
`double getmu(void)` returns the subtraction scale μ.
`double out(const int n)` returns the value of the n'th LEC.
`void out` exists in many varieties, 11 double references and a string returning all private data, 11 double references returning all LECS and the subtraction scale, 10 double references returning all LECs, 8 double references returning $l^r_1, \ldots, l^r_8$ and the subtraction scale and 8 double references returning $l^r_1, \ldots, l^r_{10}$.

`void changescale(const double newmu)`
This changes the subtraction scale to the new value given by `muin` and changes the LECs according to the running derived in [13].

Operators defined: `<<`, `>>`, $+$, $-$ and `*`.
`<<` and `>>` are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
`*` allows to multiply an `li` by a `double` in either order. The resulting value has all LECs multiplied by the value of the `double`.
$+$ and $-$ allow to add or subtract set of LECs. The resulting value of all LECs is the sum respectively the difference. A warning is printed of the scales are different.
Defined in `linf2.h`, implemented in `linf2.cc`, examples of use in `testlinf2.cc`.

6.1.5 NLO LECs: Class `libar`

```
libar(l1bar=0.,l2bar=0.,l3bar=0.,l4bar=0.,l5bar=0.,l6bar=0.,l7r=0.,h1bar=0.,
   h2bar=0.,h3r=0., mu=0.77,Name="Nameless libar")
libar(liin, mpi= 0.13957061)
const double:  l1bar,...,l6bar,l7r,h1bar,h2bar,h3r,mpi
const string:  Name
const libar:  liin
```
Protected data: `double l1bar,l2bar,l3bar,l4bar,l5bar,l6bar,l7r,h1bar,h2bar,h3r` and `string name`
Physical quantities the 10 LECs, $\bar{l}_i, \bar{h}_i$ and the two which cannot be written in the barred form (three are so-called contact terms), of two-flavour ChPT as introduced in [13].
Relevant physical case: two flavour ChPT

Input member functions:
`void setlibar(const int n, const double lin)`
`void setlibar(const double lin, const int n)`

Set the value of the LECs with index n. $n = 8, 9, 10$ correspond to $\bar{h}_1, \bar{h}_2, h_3^r$.

`setname(const string namein)` Sets the name of the set of LECs.

Output member functions:
`double out(const int n)` returns the value of the n'th LEC.
`void out` exists in many varieties, 10 double references and a string returning all private data, 10 double references returning all LECS, 7 double references returning $\bar{l}_1, \ldots, \bar{l}_6, l_7^r$.

Operators defined: `<<`, `>>`. $+$, $-$ and `*` are not defined since these operations do not make sense for the barred LECs in a simple way.
`<<` and `>>` are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
Defined in `linf2.h`, implemented in `linf2.cc`, examples of use in `testlinf2.cc`.

6.2 Three flavour ChPT

6.2.1 Class: `physmass`

```
physmass(mpiin=0.135,mkin=0.495,metain=0.548,fpiin=0.0922,muin=0.77)
mpiin,mkin,metain,fpiin,muin:  const double
```
Private data: `double mpi,mk,meta,fpi,mu`
Physical quantities: pion, kaon and eta mass, pion-decay constant and subtraction scale μ.
Relevant physical case: three flavour ChPT, isospin limit.

Input member functions:
```
void setmpi(const double mpiin=0.135)
void setmk(const double mkin=0.495)
void setmeta(const double metain=0.548)
void setfpi(const double fpiin=0.0922)
void setmu(const double muiin=0.77)
```

Output member functions exist in two varieties. Those that return all or a subset of values using references or those that return one value as the function value.
```
void out(double &mpiout, double &mkout, double &metaout, double &fpiout,
      double &muout)
double getmpi(void)
double getmk(void)
double getmeta(void)
double getfpi(void)
double getmu(void)
```

Operators defined: `<<`, `>>` and `==`.

<< and >> are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
== checks for equality within relative precision of 10^{-7}. An error will occur if any of the data members is zero.

Defined in `inputs.h`, implemented in `inputs.cc`, examples of use in `testinputs.cc`.

6.2.2 Class: `lomass`

```
lomass(mp0in=0.135, mk0in=0.495, f0in=0.090, muin=0.77)
mp0in,mk0in,f0in,muin:  const double
lomass(const quarkmass mass)
```
Private data: `double mp0,mk0,f0,mu`
Physical quantities: lowest order pion mass, lowest order kaon mass, lowest order pion-decay constant and subtraction scale μ.
Relevant physical case: three flavour ChPT, isospin limit.

The constructor from a `quarkmass` is provided such that conversions can be used.
Input member functions:
```
void setmp0(const double mp0in=0.135) void setmk0(const double mk0in=0.495)
void setf0(const double f0in=0.09) void setmu(const double muin=0.77)
```

Output member functions exist in two varieties. Those that return all or a subset of values using references or those that return one value as the function value.
```
void out(double &mp0out, double &mk0out, double &f0out, double &muout)
double getmp0(void)
double getmk0(void)
double getf0(void)
double getmu(void)
```

Operators defined: <<, >> and ==.
<< and >> are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
== checks for equality within relative precision of 10^{-7}. An error will occur if any of the data members is zero.

Defined in `inputs.h`, implemented in `inputs.cc`, examples of use in `testinputs.cc`.

6.2.3 Class: `quarkmass`

```
quarkmass(B0mhatin=0.01, B0msin=0.25, f0in=0.090, muin=0.77)
B0mhatin,B0msin,f0in,muin:  const double
quarkmass(const lomass mass)
```

Private data: `double B0mhat,B0ms,f0,mu`
Physical quantities: $B_0\hat{m}, B_0 m_s$, lowest order pion-decay constant and subtraction scale μ. The quantities $B_0\hat{m}$ and $B_0 m_s$ are the LEC B_0 [14] multiplied by the average up-down quark mass and strange quark mass respectively. These are independent of the QCD scale. The lowest order pion and kaon masses are given by $m_{\pi\,\mathrm{LO}} = \sqrt{2B_0\hat{m}}$ and $m_{K\,\mathrm{LO}} = \sqrt{B_0(\hat{m}+m_s)}$
Relevant physical case: three flavour ChPT, isospin limit.

The constructor from a `lomass` is provided such that conversions can be used.
Input member functions:

```
void setB0mhat(const double B0mhatin=0.01)
void setB0ms(const double B0msin=0.25)
void setf0(const double f0in=0.09)
void setmu(const double muin=0.77)
```

Output member functions exist in two varieties. Those that return all or a subset of values using references or those that return one value as the function value.

```
void out(double &B0mhatout, double &B0msout, double &f0out, double &muout)
double getB0mhat(void)
double getB0ms(void)
double getf0(void)
double getmu(void)
```

Operators defined: <<, >> and ==.
<< and >> are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
== checks for equality within relative precision of 10^{-7}. An error will occur if any of the data members is zero.

Defined in `inputs.h`, implemented in `inputs.cc`, examples of use in `testinputs.cc`.

6.2.4 NLO LECs: Class `Li`

```
Li(l1r=0.,l2r=0.,l3r=0.,l4r=0.,l5r=0.,l6r=0.,l7r=0.,l8r=0.,l9r=0.,l10r=0.,
   h1r=0.,h2r=0.,mu=0.77,Name="nameless Li")
const double:  l1r,...,l10r,h1r,h2r,mu
const string:  Name
```

Private data: `double L1r,L2r,L3r,L4r,L5r,L6r,L7r,L8r,L9r,L10r,H1r,H2r,mu` and `string name`
Physical quantities the 12 LECs, L_i^r, H_i^r (of which two are so-called contact terms) of three-flavour ChPT as introduced in [14] and the subtraction scale μ.
Relevant physical case: three flavour ChPT

Input member functions:
`void setli(const int n, const double lin)`
`void setli(const double lin, const int n)`
Set the value of the LECs with index n. $n = 11, 12$ correspond to $H_1^r, H_2 r$.
`setmu(const double muin)`
Sets the scale μ to the value `muin`. This does *not* change the LECs, for that use `changescale`.
`setname(const string namein)` Sets the name of the set of LECs.

Output member functions:
`double out(const int n)` returns the value of the n'th LEC.
`void out` exists in many varieties, 13 double references and a string returning all private data, 13 double references returning all LECS and the subtraction scale, 12 double references returning all LECs, 11 double references returning $L_1^r, \dots, L_{10}^r$ and the subtraction scale and 10 double references returning $L_1^r, \dots, L_{10}^r$.

`void changescale(const double newmu)`
This changes the subtraction scale to the new value given by `muin` and changes the LECs according to the running derived in [14].

Operators defined: `<<`, `>>`, $+$, $-$ and `*`.
`<<` and `>>` are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
`*` allows to multiply an `Li` by a `double` in either order. The resulting value has all LECs multiplied by the value of the `double`.
$+$ and $-$ allow to add or subtract set of LECs. The resulting value of all LECs is the sum respectively the difference. A warning is printed of the scales are different.
Extra functions:
`Li Lirandom(void)`
`Li LirandomlargeNc(void)`
`Li LirandomlargeNc2(void)`
These return a set of random NLO LECs. The values are uniformly distributed between $\pm 1/(16\pi^2)$ for `Lirandom`. `LirandomlargeNc` does the same except that it leaves L_4^r, L_6^r and L_7^r zero. `LirandomlargeNc2` does the same but L_4^r, L_6^r and L_7^r get a random value between $\pm(1/3)/(16\pi^2)$. The random numbers are generated using the system generator `rand()` so initializing using something like `srand(time(0))`. These latter functions were used in the random walks in the L_i^r in [17].
Defined in `Li.h`, implemented in `Li.cc`, examples of use in `testLi.cc`.
In the subdirectory `test` there is a file `LiCiBE14.dat` that contains the last fit of the LECs [18].

6.2.5 NNLO LECs: Class `Ci`

`Ci(Cr, mu=0.77,Name="nameless Ci")`
`Ci(mu=0.77,Name="nameless Ci")`
`const double:  mu`
`const string:  Name`
Private data: `double Cr[95]`, `mu` and `string name`
Physical quantities the 94 LECs, C_i^r (of which four are so-called contact terms) of three-flavour ChPT as introduced in [19, 20] and the subtraction scale μ. The C_i^r are the dimensionless version. Scale to the dimensionfull version with appropriate powers of F_0 but in practice normally with F_π.
Relevant physical case: three flavour ChPT

Input member functions:
`void setci(const int n, const double lin)`
`void setci(const double lin, const int n)`
Set the value of the LECs with index n.
`setmu(const double muin)`
Sets the scale μ to the value `muin`. This does *not* change the LECs, for that use `changescale`.
`setname(const string namein)` Sets the name of the set of LECs.

Output member functions:
`double out(const int n)` returns the value of the n'th LEC.
`void out` exists in many varieties, with a `double Cit[95]`, a double reference and a string returning all private data, a `double Cit[95]`, a double reference returning all LECS and the subtraction scale, and a `double Cit[95]` returning the LECs only.

`void changescale(const double newmu, Li & Liin)`
`void changescale(Li & Liin, const double newmu)`
This changes the subtraction scale to the new value given by `muin` and changes the LECs according to the running derived in [20]. Note that it changes the scale of the values of the NLO LECs L_i^r in `Liin` as well.

Operators defined: `<<`, `>>`, $+$, $-$ and `*`.
`<<` and `>>` are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
`*` allows to multiply a `Ci` by a `double` in either order. The resulting value has all LECs multiplied by the value of the `double`.
$+$ and $-$ allow to add or subtract set of LECs. The resulting value of all LECs is the sum respectively the difference. A warning is printed of the scales are different.
Extra functions:
`Ci Cirandom(void)`
`Ci CirandomlargeNc(void)`

`Ci CirandomlargeNc2(void)`
These return a set of random NNLO LECs. The values are uniformly distributed between $\pm 1/(16\pi^2)^2$ for `Cirandom`. `CirandomlargeNc` does the same except that it leaves all LECs that are not single trace terms zero. `CirandomlargeNc2` does the same but the non-single-trace terms get a LEC with a random value between $\pm(1/3)/(16\pi^2)^2$. The random numbers are generated using the system generator `rand()` so initializing using something like `srand(time(0))`. These latter functions were used in the random walks in the C_i^r in [17].
Defined in `Ci.h`, implemented in `Ci.cc`, examples of use in `testCi.cc`.

6.3 n_F flavour ChPT

6.3.1 Class: quarkmassnf

```
quarkmassnf(f0in=0.090, muin=0.77, nqin=3)
quarkmassnf(const vector<double> B0mqin,f0in=0.090, muin=0.77)
quarkmassnf(const lomassnf mass)
const double:  f0in,muin
const int:  nfin
const vector<double> B0mqin
const lomassnf mass
```

Private data: `vector<double> B0mq,double f0,mu, int nq`.
Physical quantities: $B_0 m_i$ quark masses multiplied by B_0, lowest order (pion-)decay constant and subtraction scale μ. The quantities $B_0 m_i$ are the $n_q =$ `nq` quark masses multiplied by the LEC B_0 [14], B_0 for the relevant number of quarks n_F. These are independent of the QCD scale. The lowest order charged kaon mass is given by $m_{K\,\mathrm{LO}} = \sqrt{B_0(m_u + m_s)}$
Relevant physical case: n_F flavour ChPT, possibly partially quenched where we need n_q different quark masses. The masses are referred to as 1,. . . ,nq (i.e. the counting does not start with 0).

```
void setB0mq(const double B0miin,const int i)
void setB0mq(const int i, const double B0miin=0.)
void setB0mq(const vector<double> B0mqin)
void setf0(const double f0in=0.09)
void setmu(const double muin=0.77)
```

Output member functions exist in two varieties. Those that return all or a subset of values using references or those that return one value as the function value.

```
void out(vector<double> &B0mq) const;
void out(vector<double> &B0mq, double &f0out, double &muout) const;
void out(vector<double> &B0mq, double &f0out, double &muout, int &nq) const;
int getnq(void) const;
vector<double> getB0mq(void) const;
double getB0mq(const int i) const;
double getf0(void) const;
double getmu(void) const;
```

Operators defined: `<<`, `>>` and `==`.
`<<` and `>>` are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
`==` checks for equality within relative precision of 10^{-7}. An error will occur if any of the data members is zero.

Defined in `inputsnf.h`, implemented in `inputsnf.cc`, examples of use in `testinputsnf.cc`.

6.3.2 Class: `lomassnf`

```
lomassnf(f0in=0.090, muin=0.77, nmassin=3)
lomassnf(const vector<double> massin,f0in=0.090, muin=0.77)
lomassnf(const quarkmassnf mass)
const double:  f0in,muin
const int:  nfin
const vector<double> massin
```

Private data: `vector<double> mass`,`double f0`,`mu`, `int nmass`.
Physical quantities: m_{ii} the lowest order meson masses, lowest order (pion-)decay constant and subtraction scale μ. The quantities m_{ii} are the `nmass` lowest order meson masses. They correspond to $m_{ii} = \sqrt{B_0 m_i}$ with the `nmass` quark masses multiplied by the LEC B_0 [14], B_0 for the relevant number of quarks n_F.
Relevant physical case: n_F flavour ChPT, possibly partially quenched where we need n_q different quark masses. The masses are referred to as 1,. . . ,nq (i.e. the counting does not start with 0).

```
void setmass(const double massin,const int i)
void setmass(const int i, const double massin=0.)
void setmass(const vector<double> B0mqin)
void setf0(const double f0in=0.09)
void setmu(const double muin=0.77)
```

Output member functions exist in two varieties. Those that return all or a subset of values using references or those that return one value as the function value.

```
void out(vector<double> &massout) const;
void out(vector<double> &massout, double &f0out, double &muout) const;
void out(vector<double> &massout, double &f0out, double &muout, int &nmass)
const;
int getnmass(void) const;
vector<double> getmass(void) const;
double getmass(const int i) const;
double getf0(void) const;
```

`double getmu(void) const;`

Operators defined: `<<`, `>>` and `==`.
`<<` and `>>` are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
`==` checks for equality within relative precision of 10^{-7}. An error will occur if any of the data members is zero.

Defined in `inputsnf.h`, implemented in `inputsnf.cc`, examples of use in `testinputsnf.cc`.

6.3.3 NLO LECs: Class `Linf`

```
Linf(l0r=0.,l1r=0.,l2r=0.,l3r=0.,l4r=0.,l5r=0.,l6r=0.,l7r=0.,l8r=0.,l9r=0.,
   l10r=0.,l11r=0.,h1r=0.,h2r=0.,mu=0.77,Name="nameless Linf",const int nfin=3)
const double:  l0r,...,l11r,h1r,h2r,mu
const string:  Name
const int:  nfin
```
Private data: `double L0r,L1r,L2r,L3r,L4r,L5r,L6r,L7r,L8r,L9r,L10r,L11r,H1r,H2r,mu`, `int nf` and `string name`
Physical quantities the 13 LECs, $L^r_{i=0,10}$, H^r_i (of which two are so-called contact terms) and the extra equation of motion term LEC L^r_{11} of n_F-flavour ChPT as introduced in [20] and the subtraction scale μ. The extra constant L^r_{11} is added to be able to deal with two-flavour partially quenched ChPT.
Relevant physical case: n_F flavour ChPT and partially quenched ChPT

Input member functions:
`void setnf(const int nfin)`
`void setlinf(const int n, const double lin)`
`void setlinf(const double lin, const int n)`
Sets the value of the LECs with index n. $n = 12, 13$ correspond to H^r_1, H^r_2.
`setmu(const double muin)`
Sets the scale μ to the value `muin`. This does *not* change the LECs, for that use `changescale`.
`setname(const string namein)` Sets the name of the set of LECs.

Output member functions:
`double out(const int n)` returns the value of the n'th LEC.
`void out` exists in many varieties, 15 double references, a string and an integer returning all private data, 15 double references and an integer returning all LECS and the subtraction scale and the number of flavours, 13 double references and an integer returning all LECs and n_F, 12 double and an integer references returning $L^r_1, \ldots, L^r_{11}$ and the subtraction scale and the number of flavours, 11 double and an integer references returning $L^r_1, \ldots, L^r_{11}$ and the number of flavours, and 11 double references returning $L^r_1, \ldots, L^r_{11}$.
`int getnf(void)` returns `nf`

`void changescale(const double newmu)`
This changes the subtraction scale to the new value given by `muin` and changes the LECs according to the running derived in [14].

Operators defined: `<<`, `>>`, $+$, $-$ and `*`.
`<<` and `>>` are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
`*` allows to multiply an `Li` by a `double` in either order. The resulting value has all LECs multiplied by the value of the `double`.
$+$ and $-$ allow to add or subtract set of LECs. The resulting value of all LECs is the sum respectively the difference. A warning is printed of the scales are different.
Extra functions:
`Linf Linfrandom(void)`
This returns a set of random NLO LECs. The values are uniformly distributed between $\pm 1/(16\pi^2)$. The random numbers are generated using the system generator `rand()` so initializing using something like `srand(time(0))`.

Defined in `Linf.h`, implemented in `Linf.cc`, examples of use in `testLinf.cc`.

6.3.4 NNLO LECs: Class `Ki`

`Ki(Kr, mu=0.77,Name="nameless Ci",nfin=3)`
`Ci(mu=0.77,Name="nameless Ci")`
`const double:  mu`
`const string:  Name`
`const int:  nfin`
Private data: `double Kr[116], mu, int nf` and `string name`
Physical quantities the 115 LECs, K_i^r (of which three are so-called contact terms) of n_F-flavour ChPT as introduced in [19, 20] and the subtraction scale μ. The K_i^r are the dimensionless version. Scale to the dimensionfull version with appropriate powers of F_0 but in practice normally with F_π.
Relevant physical case: n_F flavour ChPT

Input member functions:
`void setki(const int n, const double kin)`
`void setki(const double kin, const int n)`
Set the value of the LECs with index n.
`setmu(const double muin)`
Sets the scale μ to the value `muin`. This does *not* change the LECs, for that use `changescale`.
`setname(const string namein)` Sets the name of the set of LECs.

Output member functions:

`double out(const int n)` returns the value of the n'th LEC.
`void out` exists in many varieties, with a `double Kit[116]`, a double reference and a string returning all private data, a `double Kit[116]`, a double reference returning all LECS and the subtraction scale, and a `double Kit[116]` returning the LECs only.
`int getnf(void)` returns `nf`

`void changescale(const double newmu, Linf & Liin)`
`void changescale(Linf & Liin, const double newmu)`
This changes the subtraction scale to the new value given by `muin` and changes the LECs according to the running derived in [20]. Note that it changes the scale of the values of the NLO LECs L_i^r in `Liin` as well.

Operators defined: `<<`, `>>`, $+$, $-$ and `*`.
`<<` and `>>` are defined such that output and input streams work as expected. The input stream should be exactly in the format provided by the output stream.
`*` allows to multiply a `Ci` by a `double` in either order. The resulting value has all LECs multiplied by the value of the `double`.
$+$ and $-$ allow to add or subtract set of LECs. The resulting value of all LECs is the sum respectively the difference. A warning is printed of the scales are different.
Extra functions:
`Ki Kirandom(void)`
This returns a set of random NNLO LECs. The values are uniformly distributed between $\pm 1/(16\pi^2)^2$. The random numbers are generated using the system generator `rand()` so initializing using something like `srand(time(0))`.

Defined in `Ki.h`, implemented in `Ki.cc`, examples of use in `testKi.cc`.

6.4 Vectors

6.4.1 Class: `fourvector`

This defines a class that deals with a four vector or Lorentz vector. The data structure is a vector of four `double` containing the t, x, y, z components.
`fourvector(a0,a1,a2,a3)`
`fourvector(const vector<double> vv)`
Private data: `vector<double> v={0.,0.,0.,0.}`
`const double:  a0,a1,a2,a3`
`vv` must have four elements.

Input member functions:
`void set(const int n, const double xn)`
`void set(cons double xn, const int n)`
set the n-th coordinate to `xn` with $n = (0, 1, 2, 3) = (t, x, y, z)$.

Output member functions:
`double out(const int n)` returns the n-th coordinate with $n = (0, 1, 2, 3) = (t, x, y, z)$.
The conversion to a `vector<double>` is also defined, so a fourvector can be used where a `vector<double>` is expected.

Operators defined: `>>`,`<<`,`+`, `-`,`*`,`/`.
`<<` and `>>` are defined with four numbers output and input as expected. `*` allows to multiply a vector with a number from both sides. `/` allows a vector to be divided by a number. $+$ returns the sum of two fourvectors. $-$ exists in two varieties, it can return the opposite fourvector as a prefix operator or the difference of two fourvectors. `*` also allows the product of two fourvectors, it then returns the Lorentz invariant scalar product.

Defined in `fourvector.h`, implemented in `fourvector.cc`, examples of use in `testfourvector.cc`.

7 Loop integrals

Loop integrals are done with dimensional regularization and we use the standard ChPT variant of $\overline{MS}$. At one-loop order it was defined in [13, 14]. The definition at two-loop order can be found in [20].
We define for subtraction purposes:

$$d = 4 - 2\epsilon, \quad C = \ln(4\pi) + 1 - \gamma, \quad \lambda_0 = \frac{1}{\epsilon} + C, \quad \lambda_1 = \lambda_0 + C, \quad \lambda_2 = \lambda_0^2 + C^2\,. \tag{19}$$

The d-dimensional Feynman integrals do not depend directly on the subtraction scale. However, renormalization will always introduce the correct dependence. We define the one-loop integrals multiplied by an extra factor of $\mu^{2\epsilon}$ and the two-loop integrals with an extra factor of $\mu^{4\epsilon}$. This introduces the μ dependence in the expressions given below.
References are to places where the integrals are defined and/or the method used elaborated.

7.1 Tadpole or one-propagator integrals

These are defined by

$$\begin{aligned} A(n, m^2) &= \frac{\mu^{4-d}}{i}\int \frac{d^dq}{(2\pi)^d}\frac{1}{(q^2-m^2)^n}\,, \\ A(m^2,\mu^2) &= A(1,m^2)\,, \quad B(m^2,\mu^2) = A(2,m^2)\,, \quad C(m^2,\mu^2) = A(3,m^2)\,. \end{aligned} \tag{20}$$

The expansions in ϵ are given by, see e.g. [21],

$$A(m^2,\mu^2) = \frac{\lambda_0 m^2}{16\pi^2} + \overline{A}(m^2,\mu^2) + \epsilon A^\epsilon(m^2,\mu^2) + \mathcal{O}(\epsilon^2)\,,$$

$$\begin{aligned}B(m^2,\mu^2) &= \frac{\lambda_0}{16\pi^2} + \overline{B}(m^2,\mu^2) + \epsilon B^\epsilon(m^2,\mu^2) + \mathcal{O}(\epsilon^2)\,,\\ C(m^2,\mu^2) &= \overline{C}(m^2,\mu^2) + \epsilon C^\epsilon(m^2,\mu^2) + \mathcal{O}(\epsilon^2)\,.\end{aligned} \tag{21}$$

The $\mathcal{O}(\epsilon)$ terms are further expanded as

$$\begin{aligned}A^\epsilon(m^2,\mu^2) &= \frac{m^2}{16\pi^2}\left(\frac{1}{2}C^2 - C\log\frac{m^2}{\mu^2}\right) + \overline{A}^\epsilon(m^2,\mu^2)\,,\\ B^\epsilon(m^2,\mu^2) &= \frac{1}{16\pi^2}\left(\frac{1}{2}C^2 - C\log\frac{m^2}{\mu^2} - C\right) + \overline{B}^\epsilon(m^2,\mu^2)\,,\\ C^\epsilon(m^2,\mu^2) &= \frac{1}{16\pi^2}\left(-\frac{C}{2m^2}\right) + \overline{C}^\epsilon(m^2,\mu^2)\,.\end{aligned} \tag{22}$$

The analytical expressions are

$$\begin{aligned}\overline{A}(m^2,\mu^2) &= \frac{-m^2}{16\pi^2}\log\frac{m^2}{\mu^2} & \overline{A}^\epsilon(m^2,\mu^2) &= \frac{m^2}{16\pi^2}\left(\frac{1}{2} + \frac{\pi^2}{12} + \frac{1}{2}\log^2\frac{m^2}{\mu^2}\right)\,,\\ \overline{B}(m^2,\mu^2) &= \frac{1}{16\pi^2}\left(-1 - \log\frac{m^2}{\mu^2}\right) & \overline{B}^\epsilon(m^2,\mu^2) &= \frac{1}{16\pi^2}\left(\frac{1}{2} + \frac{\pi^2}{12} + \frac{1}{2}\log^2\frac{m^2}{\mu^2} + \log\frac{m^2}{\mu^2}\right)\,,\\ \overline{C}(m^2,\mu^2) &= \frac{1}{16\pi^2}\frac{-1}{2m^2} & \overline{C}^\epsilon(m^2,\mu^2) &= \frac{1}{16\pi^2}\left(\frac{1}{2m^2} + \frac{1}{2m^2}\log\frac{m^2}{\mu^2}\right)\,,\end{aligned} \tag{23}$$

`double Ab(const double msq, const double mu2)`: returns $\overline{A}(m^2,\mu^2)$.
`double Bb(const double msq, const double mu2)`: returns $\overline{B}(m^2,\mu^2)$.
`double Cb(const double msq, const double mu2)`: returns $\overline{C}(m^2,\mu^2)$.
`double Abeps(const double msq, const double mu2)`: returns $\overline{A}^\epsilon(m^2,\mu^2)$.
`double Bbeps(const double msq, const double mu2)`: returns $\overline{B}^\epsilon(m^2,\mu^2)$.
`double Cbeps(const double msq, const double mu2)`: returns $\overline{C}^\epsilon(m^2,\mu^2)$.
`double Ab(const int n,const double msq, const double mu2)`: returns $\overline{A}(m^2,\mu^2)$, $\overline{B}(m^2,\mu^2)$, $\overline{C}(m^2,\mu^2)$ for $n = 1,2,3$.

Defined in `oneloopintegrals.h`, implemented in `oneloopintegrals.cc`, examples of use in `testoneloopintegrals.cc`.

7.2 Bubbles or two-propagator integrals

7.2.1 Definitions

We first define the abbreviation

$$\langle X\rangle = \frac{\mu^{4-d}}{i}\int\frac{d^dq}{(2\pi)^d}\frac{X}{(q^2-m_1^2)\left((q-p)^2-m_2^2\right)}\,. \tag{24}$$

The bubble integrals themselves are defined by, see e.g. [22],

$$B(m_1^2,m_2^2,p^2,\mu^2) = \langle 1\rangle\,,$$

$$\begin{aligned}
B_\mu(m_1^2,m_2^2,p,\mu^2) = \langle q_\mu\rangle \qquad &= p_\mu B_1(m_1^2,m_2^2,p^2,\mu^2)\,,\\
B_{\mu\nu}(m_1^2,m_2^2,p,\mu^2) = \langle q_\mu q_\nu\rangle \qquad &= p_\mu p_\nu B_{21}(m_1^2,m_2^2,p^2,\mu^2) + g_{\mu\nu}B_{22}(m_1^2,m_2^2,p^2,\mu^2)\,,\\
B_{\mu\nu\rho}(m_1^2,m_2^2,p,\mu^2) = \langle q_\mu q_\nu q_\rho\rangle \qquad &= p_\mu p_\nu p_\rho B_{31}(m_1^2,m_2^2,p^2,\mu^2)\\
&\quad + \left(g_{\mu\nu}p_\rho + g_{\mu\rho}p_\nu + g_{\rho\nu}p_\mu\right)B_{32}(m_1^2,m_2^2,p^2,\mu^2)\,. \qquad (25)
\end{aligned}$$

The methods of [23] can be used to deduce the relations

$$\begin{aligned}
B_1(m_1^2,m_2^2,p^2,\mu^2) =& -\frac{1}{2p^2}\left(A(m_1^2,\mu^2) - A(m_2^2,\mu^2) + (m_2^2-m_1^2-p^2)B(m_1^2,m_2^2,p^2,\mu^2)\right)\,,\\
B_{22}(m_1^2,m_2^2,p^2,\mu^2) =& \frac{1}{2(d-1)}\Big(A(m_2^2,\mu^2) + 2m_1^2 B(m_1^2,m_2^2,p^2,\mu^2)\\
&+ (m_2^2-m_1^2-p^2)B_1(m_1^2,m_2^2,p^2,\mu^2)\Big)\,,\\
B_{21}(m_1^2,m_2^2,p^2,\mu^2) =& \frac{1}{p^2}\left(A(m_2^2,\mu^2) + m_1^2 B(m_1^2,m_2^2,p^2,\mu^2) - dB_{22}(m_1^2,m_2^2,p^2,\mu^2)\right)\,. \qquad (26)
\end{aligned}$$

This allows to rewrite all in terms of $B(m_1^2,m_2^2,p^2,\mu^2)$. These relations are used for the analytical evaluations given below.
The final evalaution is done by using a Feynman parameter x to combine the propagators and use the results for the tadpoles. The x integral needed can be done analytically or numerically.
The functions are then all expanded in terms of ϵ. The arguments of the various Bubble integrals are not written out.

$$\begin{aligned}
B &= \frac{\lambda_0}{16\pi^2} + \overline{B} + \epsilon B^\epsilon + \mathcal{O}(\epsilon^2)\,,\\
B_1 &= \frac{\lambda_0}{16\pi^2}\frac{1}{2} + \overline{B}_1 + \epsilon B_1^\epsilon + \mathcal{O}(\epsilon^2)\,,\\
B_{21} &= \frac{\lambda_0}{16\pi^2}\frac{1}{3} + \overline{B}_{21} + \epsilon B_{21}^\epsilon + \mathcal{O}(\epsilon^2)\,,\\
B_{22} &= \frac{\lambda_0}{16\pi^2}\left(\frac{m_1^2}{4} + \frac{m_2^2}{4} - \frac{p^2}{12}\right) + \overline{B}_{22} + \epsilon B_{22}^\epsilon + \mathcal{O}(\epsilon^2)\,,\\
B_{31} &= \frac{\lambda_0}{16\pi^2}\frac{1}{4} + \overline{B}_{31} + \epsilon B_{31}^\epsilon + \mathcal{O}(\epsilon^2)\,,\\
B_{32} &= \frac{\lambda_0}{16\pi^2}\left(\frac{m_1^2}{12} + \frac{m_2^2}{6} - \frac{p^2}{24}\right) + \overline{B}_{32} + \epsilon B_{32}^\epsilon + \mathcal{O}(\epsilon^2)\,, \qquad (27)
\end{aligned}$$

7.2.2 Analytical implementation

The functions in this section are all implemented fully analytically.

`const double:  msq, m1sq, m2sq, psq, mu2` these are $m^2, m_1^2, m_2^2, p^2, \mu^2$.
`dcomplex Bb(m1sq, m2sq, psq, mu2)`: returns $\overline{B}(m_1^2,m_2^2,p^2,\mu^2)$
`dcomplex Bb(msq, psq, mu2)`: returns $\overline{B}(m^2,m^2,p^2,\mu^2)$ using the simpler equal mass

formula.
`dcomplex B1b(m1sq, m2sq, psq, mu2)`: returns $\overline{B}_1(m_1^2, m_2^2, p^2, \mu^2)$
`dcomplex B21b(m1sq, m2sq, psq, mu2)`: returns $\overline{B}_{21}(m_1^2, m_2^2, p^2, \mu^2)$
`dcomplex B22b(m1sq, m2sq, psq, mu2)`: returns $\overline{B}_{22}(m_1^2, m_2^2, p^2, \mu^2)$
`dcomplex B22b(msq, psq, mu2)`: returns $\overline{B}_{22}(m^2, m^2, p^2, \mu^2)$ using the simpler equal mass formula.

Defined in `oneloopintegrals.h`, implemented in `oneloopintegrals.cc`, examples of use in `testoneloopintegrals.cc`.

7.2.3 Numerical implementation

The functions in this section are all implemented using a numerical complex integration over x. The integration routine used can be specified using the macro `WINTEGRAL` which defaults to `jbwgauss`. Any of the complex integration routines of `jbnumlib` can be used instead.
`const double:  m1sq, m2sq, psq, mu2` these are m_1^2, m_2^2, p^2, μ^2.
`dcomplex Bbnum(m1sq, m2sq, psq, mu2)`: returns $\overline{B}(m_1^2, m_2^2, p^2, \mu^2)$
`dcomplex B1bnum(m1sq, m2sq, psq, mu2)`: returns $\overline{B}(m_1^2, m_2^2, p^2, \mu^2)$
`dcomplex B21bnum(m1sq, m2sq, psq, mu2)`: returns $\overline{B}_{21}(m_1^2, m_2^2, p^2, \mu^2)$
`dcomplex B22bnum(m1sq, m2sq, psq, mu2)`: returns $\overline{B}_{22}(m_1^2, m_2^2, p^2, \mu^2)$
`dcomplex B31bnum(m1sq, m2sq, psq, mu2)`: returns $\overline{B}_{31}(m_1^2, m_2^2, p^2, \mu^2)$
`dcomplex B32bnum(m1sq, m2sq, psq, mu2)`: returns $\overline{B}_{32}(m_1^2, m_2^2, p^2, \mu^2)$

The precision of the numerical integration can be set and obtained:

`void setprecisiononeloopintegrals(const double eps)` sets the precison to `eps`.
`double getprecisiononeloopintegrals(void)` returns the present precision. The default is `1e-10`.

Defined in `oneloopintegrals.h`, implemented in `oneloopintegrals.cc`, examples of use in `testoneloopintegrals.cc`:

7.3 Bubbles or two-propagator integrals with different powers of propagators

7.3.1 Definitions

For partially quenched and quenched calculations we also need the bubble integrals with higher powers of the propagators. These can be derived by taking the derivatives w.r.t. to m_1^2 or m_2^2 of the definitions in Sect. 7.2. The definitions we use are

$$\langle X \rangle_k = \frac{\mu^{4-d}}{i} \int \frac{d^dq}{(2\pi)^d} \frac{X}{(q^2 - m_1^2)^{n_1} \left((q-p)^2 - m_2^2\right)^{n_2}} \,. \tag{28}$$

k indicates the values of (n_1, n_2) via

k	n_1	n_2
1	1	1
2	2	1
3	1	2
4	2	2

The bubble integrals themselves are defined by

$$\begin{aligned}
B(k,m_1^2,m_2^2,p^2,\mu^2) &= \langle 1\rangle_k\,,\\
B_\mu(k,m_1^2,m_2^2,p,\mu^2) &= \langle q_\mu\rangle_k &&= p_\mu B_1(k,m_1^2,m_2^2,p^2,\mu^2)\,,\\
B_{\mu\nu}(k,m_1^2,m_2^2,p,\mu^2) &= \langle q_\mu q_\nu\rangle_k &&= p_\mu p_\nu B_{21}(k,m_1^2,m_2^2,p^2,\mu^2) + g_{\mu\nu}B_{22}(k,m_1^2,m_2^2,p^2,\mu^2)\,,
\end{aligned} \tag{29}$$

These integrals were derived during the work on [39].

7.3.2 Analytical implementation

The arguments of the functions are `nprop` which indicates k in the definition (28).

`const double:  msq, m1sq, m2sq, psq, mu2` these are $m^2, m_1^2, m_2^2, p^2, \mu^2$.

`dcomplex Bb(nprop,m1sq, m2sq, psq, mu2)`: returns $\overline{B}(k,m_1^2,m_2^2,p^2,\mu^2)$
`dcomplex B1b(nprop,m1sq, m2sq, psq, mu2)`: returns $\overline{B}_1(k,m_1^2,m_2^2,p^2,\mu^2)$
`dcomplex B21b(nprop,m1sq, m2sq, psq, mu2)`: returns $\overline{B}_{21}(k,m_1^2,m_2^2,p^2,\mu^2)$
`dcomplex B22b(nprop,m1sq, m2sq, psq, mu2)`: returns $\overline{B}_{22}(k,m_1^2,m_2^2,p^2,\mu^2)$

At present only the cases $k = 1, 2, 3$ are implemented.
Defined in `quenchedoneloopintegrals.h`, implemented in `quenchedoneloopintegrals.cc`, examples of use in `testquenchedoneloopintegrals.cc`.

7.4 Sunset integrals

7.4.1 Definition

We first define the abbreviation

$$\langle\langle X\rangle\rangle = \left(\frac{\mu^{4-d}}{i}\right)\int \frac{d^dr}{(2\pi)^d}\frac{d^ds}{(2\pi)^d}\frac{X}{(r^2-m_1^2)\,(s^2-m_2^2)\,((r+s-p)^2-m_3^2)}\,. \tag{30}$$

The sunset integrals themselves are defined by

$$H(m_1^2,m_2^2,m_3^2,p^2,\mu^2) = \langle\langle 1\rangle\rangle\,,$$

$$\begin{aligned}
H_\mu(m_1^2,m_2^2,m_3^2,p,\mu^2) &= \langle\langle r_\mu\rangle\rangle &&= p_\mu H_1(m_1^2,m_2^2,m_3^2,p^2,\mu^2)\,,\\
H_{\mu\nu}(m_1^2,m_2^2,m_3^2,p,\mu^2) &= \langle\langle r_\mu r_\nu\rangle\rangle &&= p_\mu p_\nu H_{21}(m_1^2,m_2^2,m_3^2,p^2,\mu^2)\\
&&&\quad + g_{\mu\nu} H_{22}(m_1^2,m_2^2,m_3^2,p^2,\mu^2)\,,\\
H_{\mu\nu\rho}(m_1^2,m_2^2,m_3^2,p,\mu^2) &= \langle\langle r_\mu r_\nu r_\rho\rangle\rangle &&= p_\mu p_\nu p_\rho H_{31}(m_1^2,m_2^2,m_3^2,p^2,\mu^2)\\
&&&\quad + \left(g_{\mu\nu}p_\rho + g_{\mu\rho}p_\nu + g_{\rho\nu}p_\mu\right) H_{32}(m_1^2,m_2^2,m_3^2,p^2,\mu^2)\,.
\end{aligned} \tag{31}$$

The needed integrals with s_μ replacing some of the r_μ in the definitions can be related to those without s_μ as descibed in [21]. The evaluation of these sunsetintegrals has been done in [21]. Further references can be found there.
We extract the parts the divergent parts and the parts containing C via

$$H(m_1^2,m_2^2,m_3^2,p^2,\mu^2) = \frac{1}{(16\pi^2)^2}\Big[(\lambda_2/2)\left(m_1^2+m_2^2+m_3^2\right) + (\lambda_1/2)\big(m_1^2(1-\log(m_1^2/\mu^2))$$
$$+ m_2^2(1-\log(m_2^2/\mu^2)) + m_3^2(1-\log(m_3^2/\mu^2)) - (p^2/2)\big)\Big]$$
$$+ H^F(m_1^2,m_2^2,m_3^2,p^2,\mu^2) + \mathcal{O}(\epsilon)\,, \tag{32}$$

$$H_1(m_1^2,m_2^2,m_3^2,p^2,\mu^2) = \frac{1}{(16\pi^2)^2}\Big[(\lambda_2/4)\left(m_2^2+m_3^2\right) + (\lambda_1/8)\big(2m_1^2$$
$$+ m_2^2(1-4\log(m_2^2/\mu^2)) + m_3^2(1-4\log(m_3^2/\mu^2)) - (2p^2/3)\big)\Big]$$
$$+ H_1^F(m_1^2,m_2^2,m_3^2,p^2,\mu^2) + \mathcal{O}(\epsilon)\,, \tag{33}$$

$$H_{21}(m_1^2,m_2^2,m_3^2,p^2,\mu^2) = \frac{1}{(16\pi^2)^2}\Big[(\lambda_2/6)\left(m_2^2+m_3^2\right) + (\lambda_1/36)\big(3m_1^2$$
$$+ m_2^2(2-12\log(m_2^2/\mu^2)) + m_3^2(2-12\log(m_3^2/\mu^2))$$
$$- (3p^2/2)\big)\Big] + H_{21}^F(m_1^2,m_2^2,m_3^2,p^2,\mu^2) + \mathcal{O}(\epsilon)\,, \tag{34}$$

$$H_{31}(m_1^2,m_2^2,m_3^2,p^2,\mu^2) = \frac{1}{(16\pi^2)^2}\Big[(\lambda_2/8)\left(m_2^2+m_3^2\right) + (\lambda_1/96)\big(4m_1^2$$
$$+ m_2^2(3-24\log(m_2^2/\mu^2)) + m_3^2(3-24\log(m_3^2/\mu^2))$$
$$- (12p^2/5)\big)\Big] + H_{31}^F(m_1^2,m_2^2,m_3^2,p^2,\mu^2) + \mathcal{O}(\epsilon)\,, \tag{35}$$

The routines for the sunset integrals calculate the value at $p^2 = 0$ and the derivative there analytically. The remainder is then calculated with a rather smoot integral valid below threshold for the `double hh` functions and with a dispersive method for the `dcomplex zhh` functions. The latter is valid above and below threshold. The functions returning the derivative w.r.t. p^2 calculate the value at $p^2 = 0$ analytically and the remainder via a numerical integration as above.

7.4.2 Functions

The integration routines needed can be set using the macro `DINTEGRAL` for the real integration, default is `jbquad15`, and `SINTEGRAL` for the real integration with a singularity, default is `jbdsing15`. Any of the similar routines in `jbnumlib` can be used instead.

`const double: m1sq,m2sq,m3sq,psq,mu2`: these are $m_1^2, m_2^2, m_3^2, p^2, \mu^2$.

Valid below threshold:
`double hh(m1sq, m2sq, m3sq, psq, mu2)` returns $H^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`double hh1(m1sq, m2sq, m3sq, psq, mu2)` returns $H_1^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`double hh21(m1sq, m2sq, m3sq, psq, mu2)` returns $H_{21}^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`double hh31(m1sq, m2sq, m3sq, psq, mu2)` returns $H_{31}^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`double hhd(m1sq, m2sq, m3sq, psq, mu2)` returns $(\partial/\partial p^2)H^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`double hh1d(m1sq, m2sq, m3sq, psq, mu2)` returns $(\partial/\partial p^2)H_1^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`double hh21d(m1sq, m2sq, m3sq, psq, mu2)` returns $(\partial/\partial p^2)H_{21}^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$

Valid above and below threshold:
`dcomplex zhh(m1sq, m2sq, m3sq, psq, mu2)` returns $H^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`dcomplex zhh1(m1sq, m2sq, m3sq, psq, mu2)` returns $H_1^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`dcomplex zhh21(m1sq, m2sq, m3sq, psq, mu2)` returns $H_{21}^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`dcomplex zhh31(m1sq, m2sq, m3sq, psq, mu2)` returns $H_{31}^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`dcomplex zhhd(m1sq, m2sq, m3sq, psq, mu2)` returns $(\partial/\partial p^2)H^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`dcomplex zhh1d(m1sq, m2sq, m3sq, psq, mu2)` returns $(\partial/\partial p^2)H_1^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`dcomplex zhh21d(m1sq, m2sq, m3sq, psq, mu2)` returns $(\partial/\partial p^2)H_{21}^F(m_1^2, m_2^2, m_3^2, p^2, \mu^2)$

`void setprecisionsunsetintegrals(const double eps)` sets the precison to `eps`.
`double getprecisionsunsetintegrals(void)` returns the present precision. The default is `1e-10`.

Defined in `sunsetintegrals.h`, implemented in `sunsetintegrals.cc`, examples of use in `testsunsetintegrals.cc`:

7.5 Sunsetintegrals with different powers of propagators

7.5.1 Definition

We first define the abbreviation

$$\langle\langle X\rangle\rangle_n = \left(\frac{\mu^{4-d}}{i}\right)\int \frac{d^d r}{(2\pi)^d}\frac{d^d s}{(2\pi)^d}\frac{X}{(r^2-m_1^2)^i\,(s^2-m_2^2)^j\,((r+s-p)^2-m_3^2)^k}\,. \tag{36}$$

The translation of n to values for i, j, k is given in Tab. 1. The sunset integrals themselves are defined by

$$H(n, m_1^2, m_2^2, m_3^2, p^2, \mu^2) = \langle\langle 1\rangle\rangle_n\,,$$

n	1	2	3	4	5	6	7	8
i	1	2	1	1	2	2	1	2
j	1	1	2	1	2	1	2	2
k	1	1	1	2	1	2	2	2

Table 1: The relation between the value of n and the powers i, j, k of the three propagators.

$$\begin{aligned} H_\mu(n, m_1^2, m_2^2, m_3^2, p, \mu^2) = \langle\langle r_\mu \rangle\rangle_n &= p_\mu H_1(n, m_1^2, m_2^2, m_3^2, p^2, \mu^2)\,, \\ H_{\mu\nu}(n, m_1^2, m_2^2, m_3^2, p, \mu^2) = \langle\langle r_\mu r_\nu \rangle\rangle_n &= p_\mu p_\nu H_{21}(n, m_1^2, m_2^2, m_3^2, p^2, \mu^2) \\ &\quad + g_{\mu\nu} H_{22}(n, m_1^2, m_2^2, m_3^2, p^2, \mu^2)\,. \end{aligned} \tag{37}$$

The needed integrals with s_μ replacing some of the r_μ in the definitions can be related to those without s_μ as descibed in [24, 25, 26, 27] The evaluation of these sunsetintegrals is by the generalziation of the methods of [21]. Further references can be found there.
The divergent parts and the parts containing C via taking derivatives w.r.t. masses of (32). We thus define the functions $H_i^F(n, m_1^2, m_2^2, m_3^2, p^2, \mu^2)$ for all cases above, $i = 0(\text{blank}), 1, 21$.
The routines for the sunset integrals calculate the value at $p^2 = 0$ and the derivative there analytically. The remainder is then calculated with a rather smoot integral valid below threshold for the `double hh` functions. The functions returning the derivative w.r.t. p^2 calculate the value at $p^2 = 0$ analytically and the remainder via a numerical integration as above.
An added addition here is that case where the Kählén function

$$\lambda(m_1^2, m_2^2, m_3^2) = \sqrt{(m_1^2 - m_2^2 - m_3^2)^2 - 4m_2^2 m_3^2}$$

vanishes, is treated correctly.

7.5.2 Functions

The integration routines needed can be set using the macro `DINTEGRAL` for the real integration, default is `jbdgauss`. Any of the similar routines in `jbnumlib` can be used instead.

`const int n`: the integer n labelling the powers of the propagators as defined in Tab. 1.
`const double: m1sq,m2sq,m3sq,psq,mu2`: these are $m_1^2, m_2^2, m_3^2, p^2, \mu^2$.

Valid below threshold:
`double hh(n,m1sq, m2sq, m3sq, psq, mu2)` returns $H^F(n, m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`double hh1(n,m1sq, m2sq, m3sq, psq, mu2)` returns $H_1^F(n, m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`double hh21(n,m1sq, m2sq, m3sq, psq, mu2)` returns $H_{21}^F(n, m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`double hhd(n,m1sq, m2sq, m3sq, psq, mu2)` returns $(\partial/\partial p^2) H^F(n, m_1^2, m_2^2, m_3^2, p^2, \mu^2)$
`double hh1d(n,m1sq, m2sq, m3sq, psq, mu2)` returns $(\partial/\partial p^2) H_1^F(n, m_1^2, m_2^2, m_3^2, p^2, \mu^2)$

`double hh21d(n,m1sq, m2sq, m3sq, psq, mu2)` returns $(\partial/\partial p^2)H^F_{21}(n,m_1^2,m_2^2,m_3^2,p^2,\mu^2)$

`void setprecisionquenchedsunsetintegrals(const double eps)` sets the precison to `eps`.
`double getprecisionquenchedsunsetintegrals(void)` returns the present precision. The default is `1e-10`.

Defined in `quenchedsunsetintegrals.h`, implemented in `quenchedsunsetintegrals.cc`, examples of use in `testquenchedsunsetintegrals.cc`:

7.6 Finite volume tadpole integrals: periodic boundary conditions

7.6.1 Definitions

The methods used for these are derived in detail in [9], references to earlier literature can be found there. The integrals used here are given in the Minkowski conventions as defined in [28]. All of the integrals are available with two different methods, one using a summation over Bessel function 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.
The tadpole integrals A and $A_{\mu\nu}$ are defined as

$$\left\{\tilde{A}^V(m^2,L,\mu^2),\tilde{A}^V_{\mu\nu}(m^2,L,\mu^2)\right\} = \frac{\mu^{4-d}}{i}\int_V \frac{d^dr}{(2\pi)^d}\frac{\{1,r_\mu r_\nu\}}{(r^2-m^2)}\,. \tag{38}$$

The B tadpole integrals are the same but with a doubled propagator, C tadpoles are with a tripled propagator and D tadpoles with a quadrupled 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. The size of the spatial directions is L.
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 $\tilde{A}^V_{\mu\nu}$ are

$$\tilde{A}^V_{\mu\nu}(m^2,L,\mu^2) = g_{\mu\nu}\tilde{A}^V_{22}(m^2,L,\mu^2) + t_{\mu\nu}\tilde{A}^V_{23}(m^2,L,\mu^2)\,. \tag{39}$$

Similar definitions are relevant for the B, C, D tadpoles. In infinite volume A_{22} is related to A and A_{23} vanishes. The relations in finite volume is given by

$$d\tilde{A}^V_{22}(m^2,L,\mu^2) + 3\tilde{A}^V_{23}(m^2,L,\mu^2) = m^2\tilde{A}^V(m^2,L,\mu^2)\,. \tag{40}$$

The relations for the other cases are:

$$\begin{aligned}
d\tilde{B}^V_{22}(m^2,L,\mu^2) + 3\tilde{B}^V_{23}(m^2,L,\mu^2) &= m^2\tilde{B}^V(m^2,L,\mu^2) + \tilde{A}^V(m^2,L,\mu^2)\,,\\
d\tilde{C}^V_{22}(m^2,L,\mu^2) + 3\tilde{C}^V_{23}(m^2,L,\mu^2) &= m^2\tilde{C}^V(m^2,L,\mu^2) + \tilde{B}^V(m^2,L,\mu^2)\,,
\end{aligned}$$

$$d\tilde{D}^V_{22}(m^2,L,\mu^2)+3\tilde{D}^V_{23}(m^2,L,\mu^2)=m^2\tilde{D}^V(m^2,L,\mu^2)+\tilde{C}^V(m^2,L,\mu^2)\,. \tag{41}$$

The full integrals are now split in the infinite volume part which was defined earlier in Sect. 7.1 and the finite volume remainder as

$$\begin{aligned}
\tilde{A}^V(m^2,L,\mu^2)=&\frac{\lambda_0 m^2}{16\pi^2}+\overline{A}(m^2,\mu^2)+\overline{A}^V(m^2,L)+\epsilon\left(A^\epsilon(m^2,\mu^2)+A^{V\epsilon}(m^2,L,\mu^2)\right)+\mathcal{O}(\epsilon^2)\,,\\
\tilde{B}^V(m^2,L,\mu^2)=&\frac{\lambda_0}{16\pi^2}+\overline{B}(m^2,\mu^2)+\overline{B}^V(m^2,L)+\epsilon\left(B^\epsilon(m^2,\mu^2)+B^{V\epsilon}(m^2,L,\mu^2)\right)+\mathcal{O}(\epsilon^2)\,,\\
\tilde{C}^V(m^2,L,\mu^2)=&\overline{C}(m^2,\mu^2)+\overline{C}^V(m^2,L)+\epsilon\left(C^\epsilon(m^2,\mu^2)+C^{V\epsilon}(m^2,L,\mu^2)\right)+\mathcal{O}(\epsilon^2)\,,\\
\tilde{D}^V(m^2,L,\mu^2)=&\overline{D}(m^2,\mu^2)+\overline{D}^V(m^2,L)+\epsilon\left(D^\epsilon(m^2,\mu^2)+D^{V\epsilon}(m^2,L,\mu^2)\right)+\mathcal{O}(\epsilon^2)\,,\\
\tilde{A}^V_{22}(m^2,L,\mu^2)=&\frac{\lambda_0 m^4}{4(16\pi^2)}+\overline{A}_{22}(m^2,\mu^2)+\overline{A}^V_{22}(m^2,L)+\epsilon\left(A^\epsilon_{22}(m^2,\mu^2)+A^{V\epsilon}_{22}(m^2,L,\mu^2)\right)\\
\tilde{B}^V_{22}(m^2,L,\mu^2)=&\frac{\lambda_0 m^2}{2(16\pi^2)}+\overline{B}_{22}(m^2,\mu^2)+\overline{B}^V_{22}(m^2,L)+\epsilon\left(B^\epsilon_{22}(m^2,\mu^2)+B^{V\epsilon}_{22}(m^2,L,\mu^2)\right)\\
\tilde{C}^V_{22}(m^2,L,\mu^2)=&\frac{\lambda_0}{4(16\pi^2)}+\overline{C}_{22}(m^2,\mu^2)+\overline{C}^V_{22}(m^2,L)+\epsilon\left(C^\epsilon_{22}(m^2,\mu^2)+C^{V\epsilon}_{22}(m^2,L,\mu^2)\right)\\
\tilde{D}^V_{22}(m^2,L,\mu^2)=&\overline{D}_{22}(m^2,\mu^2)+\overline{D}^V_{22}(m^2,L)+\epsilon\left(D^\epsilon_{22}(m^2,\mu^2)+D^{V\epsilon}_{22}(m^2,L,\mu^2)\right)+\mathcal{O}(\epsilon^2)\,,\\
\tilde{A}^V_{23}(m^2,L,\mu^2)=&\overline{A}^V_{23}(m^2,\mu^2)+\epsilon A^{V\epsilon}_{22}(m^2,L)+\mathcal{O}(\epsilon^2)\,.\\
\tilde{B}^V_{23}(m^2,L,\mu^2)=&\overline{B}^V_{23}(m^2,\mu^2)+\epsilon B^{V\epsilon}_{22}(m^2,L)+\mathcal{O}(\epsilon^2)\,.\\
\tilde{C}^V_{23}(m^2,L,\mu^2)=&\overline{C}^V_{23}(m^2,\mu^2)+\epsilon C^{V\epsilon}_{22}(m^2,L)+\mathcal{O}(\epsilon^2)\,.\\
\tilde{D}^V_{23}(m^2,L,\mu^2)=&\overline{D}^V_{23}(m^2,\mu^2)+\epsilon D^{V\epsilon}_{22}(m^2,L)+\mathcal{O}(\epsilon^2)\,.
\end{aligned} \tag{42}$$

7.6.2 Functions

The integration routines needed can be set using the macro `DINTEGRAL` for the real integration, default is `jbdgauss`. Any of the similar routines in `jbnumlib` can be used instead.

`const double: msq,L` : `msq` is m^2 and `L` is the size L of the spatial dimension.

Evaluated with theta functions:
`double AbVt(msq,L)`: returns $\overline{A}^V(m^2,L)$.
`double A22bVt(msq,L)`: returns $\overline{A}^V_{22}(m^2,L)$.
`double A23bVt(msq,L)`: returns $\overline{A}^V_{23}(m^2,L)$.
`double BbVt(msq,L)`: returns $\overline{B}^V(m^2,L)$.
`double B22bVt(msq,L)`: returns $\overline{B}^V_{22}(m^2,L)$.
`double B23bVt(msq,L)`: returns $\overline{B}^V_{23}(m^2,L)$.
`double CbVt(msq,L)`: returns $\overline{C}^V(m^2,L)$.
`double C22bVt(msq,L)`: returns $\overline{C}^V_{22}(m^2,L)$.

`double C23bVt(msq,L)`: returns $\overline{C}^V_{23}(m^2,L)$.
`double DbVt(msq,L)`: returns $\overline{D}^V(m^2,L)$.
`double D22bVt(msq,L)`: returns $\overline{D}^V_{22}(m^2,L)$.
`double D23bVt(msq,L)`: returns $\overline{D}^V_{23}(m^2,L)$.

Evaluated with Bessel functions:
`double AbVb(msq,L)`: returns $\overline{A}^V(m^2,L)$.
`double A22bVb(msq,L)`: returns $\overline{A}^V_{22}(m^2,L)$.
`double A23bVb(msq,L)`: returns $\overline{A}^V_{23}(m^2,L)$.
`double BbVb(msq,L)`: returns $\overline{B}^V(m^2,L)$.
`double B22bVb(msq,L)`: returns $\overline{B}^V_{22}(m^2,L)$.
`double B23bVb(msq,L)`: returns $\overline{B}^V_{23}(m^2,L)$.
`double CbVb(msq,L)`: returns $\overline{C}^V(m^2,L)$.
`double C22bVb(msq,L)`: returns $\overline{C}^V_{22}(m^2,L)$.
`double C23bVb(msq,L)`: returns $\overline{C}^V_{23}(m^2,L)$.
`double DbVb(msq,L)`: returns $\overline{D}^V(m^2,L)$.
`double D22bVb(msq,L)`: returns $\overline{D}^V_{22}(m^2,L)$.
`double D23bVb(msq,L)`: returns $\overline{D}^V_{23}(m^2,L)$.

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 when possible with the independent Bessel function implementation done in [29].

`void setprecisionfinitevolumeoneloopt(const double Abacc=1e-10,`
`const double Bbacc=1e-9,const bool printout=true)` sets the precision for the finite volume integrals evaluated with theta function to `Abacc` for the tadpole integrals, `Bbacc` for the bubble integrals. The last variable printout is a logical variable which can be se tto true or false, default is true. Default values are those indicated.

`void setprecisionfinitevolumeoneloopb(const int maxsum=100,`
`const double Bbacc=1e-5,const bool printout=true)` sets the precision for the finite volume integrals evaluated with Bessel functions. The first argument indicates how far the sum over Bessel functions is taken. Maximum at present is 1000. The second argument gives the precision of the numerical integration for the bubble integrals.

Defined in `finitevolumeoneloopintegrals.h`, implemented in `finitevolumeoneloop-integrals.cc`, examples of use in `testfinitevolumeoneloopintegrals.cc`.

7.7 Finite volume tadpole integrals: twisted boundary conditions

7.7.1 Definitions

Finite volume also allows for different boundary conditions. The integrals for twisted boundary conditions have been derived in [35]. Twisted boundary conditions means that for a complex field Φ we use as boundary conditions

$$\Phi(x^i + L) = e^{i\theta^i}\Phi(x^i), . \tag{43}$$

This can be done with different angles θ^i for each direction and for each field. The definitions of the integrals is very similar to (38)

$$\left\{\tilde{A}^V, \tilde{A}^V_\mu, \tilde{A}^V_{\mu\nu}, \tilde{A}^V_{\mu\nu\rho}\right\} = \frac{\mu^{4-d}}{i}\int_V \frac{d^d r}{(2\pi)^d}\frac{\{1, r_\mu, r_\mu r_\nu, r_\mu r_\nu r_\rho\}}{(r^2 - m_\Phi^2)^n}\,. \tag{44}$$

except that with twisted boundary conditions the integrals with an odd number of momenta in the numerator do not vanish. The direction of the momentum r is in the direction of the propagation of the particle corresponding to Φ.
We consider the case with the three spatial directions at finite volume and with twist angles possibly different for all angles. For all of the functions above we remove the infinite volume parts defined in Sect. 7.1 and the finite volume part

$$\tilde{A}^V = \lambda_0 A^{\text{inf}} + \overline{A} + \overline{A}^V + \mathcal{O}(\epsilon)\,. \tag{45}$$

For the cases with extra powers of momenta we define in addition

$$\overline{A}^V_\mu = \overline{A}^V_{2\mu}, \qquad \overline{A}^V_{\mu\nu} = g_{\mu\nu}\overline{A}^V_{22} + \overline{A}^V_{23\mu\nu} \qquad \overline{A}^V_{\mu\nu\rho} = \overline{A}^V_{33\mu\nu\rho} \tag{46}$$

$\overline{A}^V_{2\mu}$ and $\overline{A}^V_{23\mu\nu}$ are nonzero only for spatial components.

7.7.2 Functions

We use one generic interface here for all functions
```
double AbVtwistt(const int nprop,const int ntype, const int ncomponent,
          const double msq,const double xl, const fourvector theta)
```
`nprop` is the power n of the propagator in (44).
`ntype` labels which of the functions it returns. It can have the values:

`ntype`	returns
0	$\overline{A}^V$
2	component of $\overline{A}^V_2$
22	$\overline{A}^V_{22}$
23	component of $\overline{A}^V_{23}$
33	component of $\overline{A}^V_{33}$

`ncomponent` is ignored for `ntype` $= 0, 22$. For `ntype=2,23,33` it indicates which are the value of μ, ν, ρ. For `ntype=2` allowed values are `ncomponent=0,1,2,3` and for `ntype=23` allowed values are `ncomponent=00,01,10,02,20,30,03,11,12,21,13,31,22,23,32,33`. For `ntype=33` allowed values are `ncomponent=(100 i+10 j+k)` with `ijk=`
`000,001,010,002,020,030,003,011,012,021,013,031,022,023,032,033,`
`100,101,110,102,120,130,103,111,112,121,113,131,122,123,132,133,`
`200,201,210,202,220,230,203,211,212,221,213,231,222,223,232,233,`
`300,301,310,302,320,330,303,311,312,321,313,331,322,323,332,333`.
The convention is such that it returns the values for μ, ν, ρ as *upper* indices. The interpretation as written above[1] is needed since a literally specified number with leading zeroes is interpreted by C++ compilers as octal, not however when it is read in from a file or a stream.

`msq` is m^2 of (44).
`fourvector theta` should contain the values of the twisting angle θ^i for the three spatial directions in its three spatial components. The time component is ignored.

The integration routine needed can be set using the macro `DINTEGRAL` for the real integration, default is `jbdgauss2`. Any of the similar routines in `jbnumlib` can be used instead.

The precision of the numerical integration can be set with
`void setprecisionAbVtwistt(const double eps=1e-10)`.
The present precision can be obtained from
`double getprecisionAbVtwistt(void)`. Default value of the precision in `1e-10`.

Defined in `finitevolumeonelooptwist.h`, implemented in `finitevolumeonelooptwist.cc`, examples of use in `testfinitevolumeonelooptwist.cc`.

7.8 Finite volume bubble integrals: periodic boundary conditions

7.8.1 Definitions

The methods used for these are derived in detail in [9], references to earlier literature can be found there. The integrals used here are given in the Minkowski conventions as defined in [28]. All of the integrals are available with two different methods, one using a summation over Bessel function 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.

[1]This comment only matters in practice for three or more indices.

The bubble integrals B, B_μ and $B_{\mu\nu}$ are defined as

$$\left\{\tilde{B}^V, \tilde{B}^V_\mu, \tilde{B}^V_{\mu\nu}\right\}(m_1^2, m_2^2, p, L, \mu^2) = \frac{\mu^{4-d}}{i}\int_V \frac{d^d r}{(2\pi)^d}\frac{\{1, r_\mu, r_\mu r_\nu\}}{(r^2-m_1^2)^{n_1}((p-r)^2-m_2^2)^{n_2}}\,. \quad (47)$$

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. The size of the spatial directions is L.
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. In addition, the functions can in principle depend on the components of the vector p as well, not only via p^2.
The functions themselves are split in the infinite volume part discussed in Sect. 7.2 and a finite volume part via

$$\tilde{B}^V = \frac{\lambda_0}{16\pi^2} + \overline{B}(m^2, \mu^2) + \overline{B}^V(m^2, L) + \epsilon\left(B^\epsilon + B^{V\epsilon}\right) + \mathcal{O}(\epsilon^2)\,, \quad (48)$$

where all functions have as argument $(m_1^2, m_2^2, p, L, \mu^2)$.

7.8.2 Functions

The integration routines needed can be set using the macro `DINTEGRAL` for the real integration, default is `jbdgauss`.

`void setprecisionfinitevolumeoneloopt(const double Abacc=1e-10, const double Bbacc=1e-9,const bool printout=true)` sets the precision for the finite volume integrals evaluated with theta function to `Abacc` for the tadpole integrals, `Bbacc` for the bubble integrals. The last variable printout is a logical variable which can be set to true or false, default is false. Default values are those indicated.

`void setprecisionfinitevolumeoneloopb(const int maxsum=100, const double Bbacc=1e-5,const bool printout=true)` sets the precision for the finite volume integrals evaluated with Bessel functions. The first argument indicates how far the sum over Bessel functions is taken. Maximum at present is 1000. The second argument gives the precision of the numerical integration for the bubble integrals.

Defined in `finitevolumeoneloopintegrals.h`, implemented in `finitevolumeoneloop-integrals.cc`, examples of use in `testfinitevolumeoneloopintegrals.cc`.

7.8.2.1 $p = 0$ and periodic boundary conditions

`const double: m1sq,m2sq,L`. These correspond to m_1^2, m_2^2, L. L is the size of the finite dimensions.

Evaluated with theta functions:
`double BbVt(m1sq,m2sq,L)`: returns $\overline{B}^V(m_1^2, m_2^2, p=0, L, \mu^2)$
`double BbVt(msq,L)`: returns $\overline{B}^V(m^2, m^2, p=0, L)$

Evaluated with Bessel functions:
`double BbVb(m1sq,m2sq,L)`: return $\overline{B}^V(m_1^2, m_2^2, p=0, L, \mu^2)$
`double BbVb(msq,L)`: returns $\overline{B}^V(m^2, m^2, p=0, L, \mu^2)$

7.9 Finite volume bubble integrals: twisted boundary conditions

7.9.1 Definitions

Finite volume also allows for different boundary conditions. The integrals for twisted boundary conditions have been derived in [35]. Twisted boundary conditions means that for a complex field Φ_a we use as boundary conditions

$$\Phi(x^i + L) = e^{i\theta^i_\Phi}\Phi(x^i), . \tag{49}$$

This can be done with different angles θ^i_a for each direction and for each field. The definitions of the integrals is similar to (47)

$$\left\{\tilde{B}^V, \tilde{B}^V_\mu, \tilde{B}^V_{\mu\nu}\right\} = \frac{\mu^{4-d}}{i}\int_V \frac{d^dr}{(2\pi)^d}\frac{\{1, r_\mu, r_\mu r_\nu\}}{(r^2 - m^2_{\Phi_1})^{n_1}((p-r)^2 - m^2_{\Phi_2})^{n_2}}\,. \tag{50}$$

The direction of the momentum r is in the direction of the propagation of the particle corresponding to Φ_1. The direction of the momentum $p-r$ is in the direction of the propagation of the particle corresponding to Φ_2. This choice is made to have the interchange of Φ_1 and Φ_2 work as expected. As explained in [36], the external momenta p must have spatial components of the form

$$p^i = \left(2\pi n + \theta^i_{\Phi_1} + \theta^i_{\Phi_2}\right)\,. \tag{51}$$

This way the momentum $p-r$ satisfies the boundary conditions for Φ_2.
We consider the case with the three spatial directions at finite volume and with twist angles possibly different for all directions. For all of the functions above we remove the infinite volume parts defined in Sect. 7.2 and the finite volume part

$$\tilde{B}^V = \lambda_0 B^{\text{inf}} + \overline{B} + \overline{B}^V + \mathcal{O}(\epsilon)\,. \tag{52}$$

For the cases with extra powers of momenta we define in addition

$$\overline{B}_\mu = p_\mu \overline{B}^V_1 + \overline{B}^V_{2\mu}, \qquad \overline{B}^V_{\mu\nu} = p_\mu p_\nu \overline{B}^V_1 + g_{\mu\nu}\overline{B}^V_{22} + \overline{B}^V_{23\mu\nu}\,. \tag{53}$$

$\overline{B}^V_{2\mu}$ and $\overline{B}^V_{23\mu\nu}$ are nonzero only for spatial components.

7.9.2 Functions

We use one generic interface here for all functions

```
double BbVtwistt(const int nprop,const int ntype, const int ncomponent,
          const double m1sq, const double m2sq, const double psq,
          const double xl, const fourvector theta1, const fourvector p)
```

nprop indicates the powers n_1, n_2 of the propagators in (50) via

nprop	n_1	n_2
1	1	1
2	2	1
3	1	2
4	2	2

ntype labels which of the functions it returns. It can have the values:

ntype	returns
0	$\overline{B}^V$
1	$\overline{B}_1^V$
2	component of $\overline{B}_2^V$
21	$\overline{B}_{22}^V$
22	$\overline{B}_{22}^V$
23	component of $\overline{B}_{23}^V$

ncomponent is ignored for ntype $= 0, 1, 21, 22$. For ntype=2,23 it indicates which are the value of μ, ν. For ntype=2 allowed values are ncomponent=0,1,2,3 and for ntype=23 allowed values are ncomponent=00,01,10,02,20,30,03,11,12,21,13,31,22,23,32,33.
m1sq,m2sq are the masses squared of Φ_1, Φ_2 and psq is p^2.
fourvector theta1 should contain the values of the twisting angle $\theta^i_{\Phi_1}$ of the particle in the first propagator for the three spatial directions in its three spatial components. The time component is ignored.
fourvector p should contain the values of the four momentum p. No checks are done that it satisfies the relation (51) nor that p=psq. The value of the twist angles for Φ_2 are (implicitly) deduced from (51) in the calculation.
The precision of the numerical integration can be set with
`void setprecisionBbVtwistt(const double releps=1e-10).`
The present precision can be obtained from
`double getprecisionBbVtwistt(void).` Default value of the precision in 1e-10.
The precision of the tadpole and bubble cases can be set at the same time using

```
void setprecisionfinitevolumeonelooptwistt(const double Aeps=1e-10,
          const double Beps=1e-10, const bool printout=true)
```

Aeps for the tadpoles and Beps for the bubbles.

Defined in `finitevolumeonelooptwist.h`, implemented in `finitevolumeonelooptwist.cc`, examples of use in `testfinitevolumeonelooptwist.cc`.

7.10 Finite volume sunsetintegrals

7.10.1 Definitions

The sunset integrals are defined with

$$\langle\langle X\rangle\rangle_V = \frac{\mu^{8-2d}}{i^2}\int_V \frac{d^dr}{(2\pi)^d}\frac{d^d1}{(2\pi)^d}\frac{\{1, r_\mu, r_\mu r_\nu\}}{(r^2-m_1^2)\,(s^2-m_2^2)\,((r+s-p)^2-m_3^2)}\,. \tag{54}$$

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 [21] and of Sect. 7.4. Integrals with the other momentum s in the numerator are related using the relations shown in [21] which remain valid at finite volume in the cms frame [9].
In the cms frame we define the functions[2]

$$\begin{aligned}\tilde H^V_\mu =&\langle\langle X\rangle\rangle_V \\ \tilde H^V_\mu =&\langle\langle r_\mu\rangle\rangle_V = p_\mu \tilde H^V_1 \\ \tilde H^V_{\mu\nu} =&\langle\langle r_\mu r_\nu\rangle\rangle_V = p_\mu p_\nu \tilde H^V_{21} + g_{\mu\nu}\tilde H^V_{22} + t_{\mu\nu}\tilde H^V_{27}\,.\end{aligned} \tag{55}$$

The arguments of all functions in the cms frame are $(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$. These functions satisfy in finite volume [9],

$$\begin{aligned}\tilde H^V_1 + \tilde H^V_1(m_2^2, m_3^2, m_1^2, p^2, L, \mu^2) + \tilde H^V_1(m_3^2, m_1^2, m_2^2, p^2, L, \mu^2) =&\tilde H^V\,, \\ p^2\tilde H^V_{21} + d\tilde H^V_{22} + 3\tilde H^V_{27} - m_1^2 H =&\tilde A^V(m_2^2)\tilde A^V(m_3^2)\,.\end{aligned} \tag{56}$$

The arguments of the sunset functions in the relations, if not mentioned explicitly, are $(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
We split the functions in an infinite volume part, H_i, and a finite volume correction, H^V_i, with $\tilde H^V_i = H_i + H^V_i$. The infinite volume part has been discussed above. For the finite volume parts we define

$$\begin{aligned}H^V =&\frac{\lambda_0}{16\pi^2}\left(\overline{A}^V(m_1^2) + \overline{A}^V(m_2^2) + \overline{A}^V(m_3^2)\right) + \frac{1}{16\pi^2}\left(A^{V\epsilon}(m_1^2) + A^{V\epsilon}(m_2^2) + A^{V\epsilon}(m_3^2)\right) \\ &+ H^{VF} + \mathcal{O}(\epsilon)\,, \\ H^V_1 =&\frac{\lambda_0}{16\pi^2}\frac{1}{2}\left(\overline{A}^V(m_2^2) + \overline{A}^V(m_3^2)\right) + \frac{1}{16\pi^2}\frac{1}{2}\left(A^{V\epsilon}(m_2^2) + A^{V\epsilon}(m_3^2)\right) + H^{VF}_1 + =(\epsilon)\,, \\ H^V_{21} =&\frac{\lambda_0}{16\pi^2}\frac{1}{3}\left(\overline{A}^V(m_2^2) + \overline{A}^V(m_3^2)\right) + \frac{1}{16\pi^2}\frac{1}{3}\left(A^{V\epsilon}(m_2^2) + A^{V\epsilon}(m_3^2)\right) + H^{VF}_{21} + \mathcal{O}(\epsilon)\,,\end{aligned}$$

[2]In the cms frame, after the reduction to four dimensions, $t_{\mu\nu} = g_{\mu\nu} - p_\mu p_\nu/p^2$ but the separation appears naturally in the calculation [9]. In addition, it avoids singularities in the limit $p \to 0$.

$$H^V_{27} = \frac{\lambda_0}{16\pi^2}\left(\overline{A}^V_{23}(m_1^2) + \frac{1}{3}\overline{A}_{23}(m_2^2) + \frac{1}{3}\overline{A}^V_{23}(m_3^2)\right)$$
$$+ \frac{1}{16\pi^2}\left(A^{V\epsilon}_{23}(m_1^2) + \frac{1}{3}A^{V\epsilon}_{23}(m_2^2) + \frac{1}{3}A^{V\epsilon}_{23}(m_3^2)\right) + H^{VF}_{27} + \mathcal{O}(\epsilon)\,. \quad (57)$$

The finite parts are defined differently from the infinite volume case in [21]. The parts with $A^{V\epsilon}$ are removed here as well.
The functions H_i^{VF} can be computed with the methods of [9]. They are obtained by adding the parts labeled with G and H in Sect. 4.3 and the part of Sect. 4.4 in [9]. The derivatives w.r.t. p^2 can be treated using a simple adaptation of that method.
The method for evaluation works only below threshold. The numerical evaluation is rather slow. Playing with the precision settings for the specific case you need is very strongly recommended.

7.10.2 Functions

`const double: m1sq,m2sq,m3sq,psq,L,mu2`. These correspond to $m_1^2, m_2^2, m_3^2, p^2, L, \mu^2$.

Evaluation using theta functions:
`double hhVt(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh1Vt(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_1^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh21Vt(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_{21}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh22Vt(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_{22}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh27Vt(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_{27}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hhdVt(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh1dVt(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_1^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh21dVt(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_{21}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh22dVt(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_{22}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh27dVt(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_{27}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.

Evaluation using Bessel functions:
`double hhVb(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh1Vb(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_1^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh21Vb(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_{21}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh22Vb(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_{22}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh27Vb(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_{27}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hhdVb(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh1dVb(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_1^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh21dVb(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_{21}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh22dVb(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_{22}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh27dVb(m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_{27}^{VF}(m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.

For all cases discussed both methods, via Bessel or (generalized) Jacobi theta functions, give the same results. The derivatives w.r.t. p^2 for all the integrals were 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.

```
void setprecisionfinitevolumesunsett(const double racc=1e-5,
  const double rsacc=1e-4,const bool printout=true)
```
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 bool variable `printout` defaults to `true` and sets whether the setting is printed.

```
void setprecisionfinitevolumesunsetb(const int maxsum1=100,
   const int maxsum2=40,racc=1e-5,rsacc=1e-4,printout=true)
```
The integers `maxsum1` and `maxsum2` give how far the sum over Bessel functions is used for the case with one or two loop momenta “feeling” the finite volume. The first is maximum 1000, the second maximum 40 in the present implementation. In the latter case a triple sum is needed, hence the much lower upper bound. The double values `sunsetracc` and `sunsetrsacc` set the accuracies of the numerical integration which is still needed after the sum for both cases.

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`.

Defined in `finitevolumesunsetintegrals.h`, implemented in `finitevolumesunsetintegrals.cc` and examples of use in `testfinitevolumesunsetintegrals.cc`

7.11 Finite volume sunsetintegrals with different powers of propagators

7.11.1 Definitions

The sunset integrals with different powers of momenta are defined with

$$\langle\langle X\rangle\rangle_{nV} = \frac{\mu^{8-2d}}{i^2}\int_V \frac{d^dr}{(2\pi)^d}\frac{d^d1}{(2\pi)^d}\frac{\{1, r_\mu, r_\mu r_\nu\}}{(r^2-m_1^2)^i\,(s^2-m_2^2)^j\,((r+s-p)^2-m_3^2)^k}\,. \qquad (58)$$

The subscript V indicates that the spatial dimensions are a discrete sum rather than an integral. The translation of n to the powers i, j, k is given in Tab. 1.

In the cms frame we define the functions[3]

$$\begin{aligned}\tilde{H}^V_\mu =&\langle\langle X\rangle\rangle_{nV} \\ \tilde{H}^V_\mu =&\langle\langle r_\mu\rangle\rangle_{nV} = p_\mu \tilde{H}^V_1 \\ \tilde{H}^V_{\mu\nu} =&\langle\langle r_\mu r_\nu\rangle\rangle_{nV} = p_\mu p_\nu \tilde{H}^V_{21} + g_{\mu\nu}\tilde{H}^V_{22} + t_{\mu\nu}\tilde{H}^V_{27}\,.\end{aligned} \tag{59}$$

The arguments of all functions in the cms frame are $(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
We split the functions in an infinite volume part, H_i, and a finite volume correction, H_i^V, with $\tilde{H}_i^V = H_i + H_i^V$. The infinite volume part has been discussed above. Note that the functions in the section are defined with the derivative w.r.t. m_l^2 for the propagators with mass m_l^2. For the finite volume parts we define the subtraction as before in (57), i.e. we subtract the $A^{V\epsilon}, B^{V\epsilon}$ parts.
The functions H_i^{VF} can be computed with the methods of [9]. They are obtained by adding the parts labeled with G and H in Sect. 4.3 and the part of Sect. 4.4 in [9]. The derivatives w.r.t. p^2 can be treated using a simple adaptation of that method.
The method for evaluation works only below threshold. The numerical evaluation is rather slow. Playing with the precision settings for the specific case you need is very strongly recommended.

7.11.2 Functions

`const int n` The propagator cases as given in Tab. 1.
`const double: m1sq,m2sq,m3sq,psq,L,mu2`. These correspond to $m_1^2, m_2^2, m_3^2, p^2, L, \mu^2$.

Evaluation using theta functions:
`double hhVt(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh1Vt(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_1^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh21Vt(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_{21}^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh22Vt(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_{22}^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh27Vt(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_{27}^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hhdVt(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh1dVt(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_1^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh21dVt(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_{21}^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh22dVt(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_{22}^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh27dVt(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H_{27}^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.

Evaluation using Bessel functions:
`double hhVb(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh1Vb(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_1^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh21Vb(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_{21}^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.
`double hh22Vb(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H_{22}^{VF}(n, m_1^2, m_2^2, m_3^2, p^2, L, \mu^2)$.

[3]In the cms frame $t_{\mu\nu} = g_{\mu\nu} - p_\mu p_\nu/p^2$ but the separation appears naturally in the calculation [9]. In addition, it avoids singularities in the limit $p \to 0$.

`double hh27Vb(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $H^{VF}_{27}(n,m_1^2,m_2^2,m_3^2,p^2,L,\mu^2)$.
`double hhdVb(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H^{VF}(n,m_1^2,m_2^2,m_3^2,p^2,L,\mu^2)$.
`double hh1dVb(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H^{VF}_{1}(n,m_1^2,m_2^2,m_3^2,p^2,L,\mu^2)$.
`double hh21dVb(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H^{VF}_{21}(n,m_1^2,m_2^2,m_3^2,p^2,L,\mu^2)$.
`double hh22dVb(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H^{VF}_{22}(n,m_1^2,m_2^2,m_3^2,p^2,L,\mu^2)$.
`double hh27dVb(n,m1sq,m2sq,m3sq,psq,L,mu2)`: returns $(\partial/\partial p^2)H^{VF}_{27}(n,m_1^2,m_2^2,m_3^2,p^2,L,\mu^2)$.

For all cases discussed both methods, via Bessel or (generalized) Jacobi theta functions, give the same results. The derivatives w.r.t. p^2 for all the integrals were 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.

```
void setprecisionfinitevolumesunsett(const double racc=1e-5,
  const double rsacc=1e-4,const bool printout=true)
```
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 bool variable `printout` defaults to `true` and sets whether the setting is printed.

```
void setprecisionfinitevolumesunsetb(const int maxsum1=100,
   const int maxsum2=40,racc=1e-5,rsacc=1e-4,printout=true)
```
The integers `maxsum1` and `maxsum2` give how far the sum over Bessel functions is used for the case with one or two loop momenta "feeling" the finite volume. The first is maximum 1000, the second maximum 40 in the present implementation. In the latter case a triple sum is needed, hence the much lower upper bound. The double values `sunsetracc` and `sunsetrsacc` set the accuracies of the numerical integration which is still needed after the sum for both cases.

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`.

Defined in `finitevolumesunsetintegrals.h`, implemented in `finitevolumesunsetintegrals.cc` and examples of use in `testfinitevolumesunsetintegrals.cc`

8 Two flavour isospin conserving results

8.1 Mass, decay constant and vacuum-expectation-value: in physical

The exansions in this subsection are defined in terms of the physical mass, m_π, and the physical pion decay constant F_π.

8.1.1 Mass

The pion mass at two-loops in two flavour ChPT was evaluated in [43, 42, 40, 41], one-loop in [13]. The expressions as given here are in [44].
The expression for the physical mass is given by

$$m_\pi^2 = m_{\pi\,0}^2 \left(1 + m_\pi^{2(4)}\right) . \tag{60}$$

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

$$m_{\pi\,0}^2 = 2B\hat{m}\,. \tag{61}$$

The higher order contributions are split in the parts depending on the NLO LECs l_i^r, on the NNLO LECs c_i^r and the remainder as

$$m_\pi^{2(4)} = m_{\pi\,l}^{2(4)} + m_{\pi\,r}^{2(4)} , \tag{62}$$

Note that when combining these with results from other sources one should be sure to use a compatible LO and NLO.

`double mpi4nf2(physmassnf2,libar)` returns $m_\pi^{2(4)}$
`double mpi4nf2(physmassnf2,li)` returns $m_\pi^{2(4)}$
`double mpi4lnf2(physmassnf2,li)` returns $m_{\pi\,l}^{2(4)}$
`double mpi4rnf2(physmassnf2)` returns $m_{\pi\,r}^{2(4)}$

The functions are defined in `massesdecayvevnf2.h`, implemented in `massesdecayvevnf2.cc` and examples of use are in `testmassdecayvevnf2.cc`.

8.1.2 Decay constant

The pion decay constant at two-loops in two flavour ChPT was evaluated in [43, 42, 40, 41], one-loop in [13]. The expressions as given here are in [44].
The expression for the physical decay constant is given by

$$F_\pi = F\left(1 + F_\pi^{(4)}\right) . \tag{63}$$

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

The higher order contributions are split in the parts depending on the NLO LECs l_i^r, on the NNLO LECs c_i^r and the remainder as

$$F_\pi^{(4)} = F_{\pi\, l}^{(4)} + F_{\pi\, r}^{(4)}\,, \tag{64}$$

Note that when combining these with results from other sources one should be sure to use a compatible LO and NLO.

`double fpi4nf2(physmassnf2,libar)` returns $F_\pi^{(4)}$
`double fpi4nf2(physmassnf2,li)` returns $F_\pi^{(4)}$
`double fpi4lnf2(physmassnf2,li)` returns $F_{\pi\, l}^{2(4)}$
`double fpi4rnf2(physmassnf2)` returns $F_{\pi\, r}^{(4)}$

The functions are defined in `massesdecayvevnf2.h`, implemented in `massesdecayvevnf2.cc` and examples of use are in `testmassdecayvevnf2.cc`.

8.1.3 vacuum-expectation-value

The vacuum-expectation-value $\langle 0|\bar{u}u|0\rangle \equiv \langle\bar{u}u\rangle$ was evaluated at one-loop in [13] in two-flavour ChPT.
The expression for the vacuum-expectation-value is given by

$$\langle\bar{u}u\rangle = -BF^2\left(1 + \langle\bar{u}u\rangle^{(4)}\right)\,. \tag{65}$$

The superscripts indicate the order of the diagrams in p that each contribution comes from. The higher order contributions are split in the parts depending on the NLO LECs l_i^r, on the NNLO LECs c_i^r and the remainder as

$$\langle\bar{u}u\rangle^{(4)} = \langle\bar{u}u\rangle_l^{(4)} + \langle\bar{u}u\rangle_r^{(4)}\,, \tag{66}$$

Note that when combining these with results from other sources one should be sure to use a compatible LO and NLO.

`double qqup4nf2(physmassnf2,libar)` returns $\langle\bar{u}u\rangle^{(4)}$
`double qqup4nf2(physmassnf2,li)` returns $\langle\bar{u}u\rangle^{(4)}$
`double qqup4lnf2(physmassnf2,li)` returns $\langle\bar{u}u\rangle_l^{2(4)}$
`double qqup4rnf2(physmassnf2)` returns $\langle\bar{u}u\rangle_r^{(4)}$

The functions are defined in `massesdecayvevnf2.h`, implemented in `massesdecayvevnf2.cc` and examples of use are in `testmassdecayvevnf2.cc`.

8.2 Mass, decay constant and vacuum-expectation-value at finite volume: in physical

The expressions treated in this section have been derived in [28] and the one-loop results in [45]. The pion mass at two-loops was done earlier in [46]. 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 `6rnf2V`, once and store them. They only depend nontrivially on the masses and size of the finite volume. The decay constant dependence is very simple, an overall factor at each order, and there is no 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. $\vec{p} = 0$.

8.2.1 Mass at finite volume: in physical

The finite volume corrections to the masses squared[4] are defined as the difference of the mass squared in finite volume and in infinite volume:

$$\begin{aligned} \Delta^V m_\pi^2 &= m_\pi^{2V} - m_\pi^{2\,V=\infty} = m_\pi^{2V(4)} + m_\pi^{2V(6)}\,, \\ m_\pi^{2V(6)} &= m_{\pi\,l}^{2V(6)} + m_{pi\,r}^{2V(6)}\,. \end{aligned} \tag{67}$$

$m_{\pi\,l}^{2V(6)}$ is the part depending on the NLO LECs l_i^r.

Pion mass (theta function method):
`double mpi4nf2Vt(const physmassnf2 massin,const double L)` returns $m_\pi^{2V(4)}$.
`double mpi6nf2Vt(const physmassnf2 massin, const li liiin, const double L)` returns $m_\pi^{2V(6)}$.
`double mpi6lnf2Vt(const physmassnf2 massin, const li liiin, const double L)` returns $m_{\pi\,l}^{2V(6)}$.
`double mpi6rnf2Vt(const physmassnf2 massin, const double L)` returns $m_{\pi\,r}^{2V(6)}$.

Pion mass (Bessel function method):
`double mpi4nf2Vb(const physmassnf2 massin,const double L)` returns $m_\pi^{2V(4)}$.
`double mpi6nf2Vb(const physmassnf2 massin, const li liiin, const double L)` returns $m_\pi^{2V(6)}$.
`double mpi6lnf2Vb(const physmassnf2 massin, const li liiin, const double L)` returns $m_{\pi\,l}^{2V(6)}$.
`double mpi6rnf2Vb(const physmassnf2 massin, const double L)` returns $m_{\pi\,r}^{2V(6)}$.

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

All these are defined in `massdecayvevnf2V.h` and implemented in `massdecayvevnf2V.h`. Examples of use are in `testmassdecayvevnf2V.cc`.

8.2.2 Decay constant at finite volume: in physical

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

$$\begin{aligned} \Delta^V F_\pi &= F_\pi^V - F_\pi^{V=\infty} = F_\pi^{V(4)} + F_\pi^{V(6)}\,. \\ F_\pi^{V(6)} &= F_{\pi\,l}^{V(6)} + F_{\pi\,r}^{V(6)}\,. \end{aligned} \tag{68}$$

Note that the correction is defined to the value of the decay constant, not divided by the the lowest order decay constant as in (73).

Pion decay constant (theta function method):
`double fpi4nf2Vt(const physmassnf2 massin,const double L)` returns $F_\pi^{V(4)}$.
`double fpi6nf2Vt(const physmassnf2 massin,const li liin,const double L)` returns $F_\pi^{V(6)}$.
`double fpi6lnf2Vt(const physmassnf2 massin,const li liin,const double L)` returns $F_{\pi\,l}^{V(6)}$.
`double fpi6rnf2Vt(const physmassnf2 massin,const double L)` returns $F_{\pi\,r}^{V(6)}$.

Pion decay constant (Bessel function method):
`double fpi4nf2Vb(const physmassnf2 massin,const double L)` returns $F_\pi^{V(4)}$.
`double fpi6nf2Vb(const physmassnf2 massin,const li liin,const double L)` returns $F_\pi^{V(6)}$.
`double fpi6lnf2Vb(const physmassnf2 massin,const li liin,const double L)` returns $F_{\pi\,l}^{V(6)}$.
`double fpi6rnf2Vb(const physmassnf2 massin,const double L)` returns $F_{\pi\,r}^{V(6)}$.

All these are defined in `massdecayvevnf2V.h` and implemented in `massdecayvevnf2V.h`. Examples of use are in `testmassdecayvevnf2V.cc`.

8.2.3 Vacuum-expectation-value at finite volume: in physical

The finite volume corrections to the vev $\langle \bar u u\rangle$ are defined as the difference of the decay constant in finite volume and in infinite volume:

$$\Delta^V \langle \bar u u\rangle = \langle \bar u u\rangle^V - \langle \bar u u\rangle^{V=\infty} = \langle \bar u u\rangle^{V=\infty}\left(1 + \langle \bar u u\rangle_\pi^{V(4)}\right)\,. \tag{69}$$

Note that the correction is defined relative to the value in infinite volume not to the lowest order value.

$\langle \bar{u}u \rangle$ (theta function method):
`double qqup4nf2Vt(const physmassnf2 massin,const double L)` returns $\langle \bar{u}u \rangle_\pi^{V(4)}$.

$\langle \bar{u}u \rangle$ (Bessel function method):
`double qqup4nf2Vb(const physmassnf2 massin,const double L)` returns $\langle \bar{u}u \rangle_\pi^{V(4)}$.

All these are defined in `massdecayvevnf2V.h` and implemented in `massdecayvevnf2V.h`. Examples of use are in `testmassdecayvevnf2V.cc`.

9 Three flavour isospin conserving results

9.1 Masses, decay constants and vacuum-expectation-values: in physical

The exansions in this subsection are defined in terms of the physical masses, m_π, m_K, m_η and the physical pion decay constant F_π.

9.1.1 Masses

The masses of the pion, kaon and eta at two-loops in three flavour ChPT were evaluated in [21]. The pion and eta mass were done earlier with a different subtraction scheme and a different way to perform the sunset integrals in [30].
The expressions for the physical masses for $a = \pi, K, \eta$ are given by

$$m^2_{a\,\mathrm{phys}} = m^2_{a\,0} + m_a^{2(4)} + m_a^{2(6)}\,. \tag{70}$$

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

$$m^2_{\pi\,0} = 2B_0\hat{m}\,, \quad m^2_{K\,0} = B_0\left(\hat{m} + m_s\right)\,, \quad m^2_{\eta\,0} = \frac{2}{3}\left(\hat{m} + 2m_2\right)\,. \tag{71}$$

The higher order contributions are split in the parts depending on the NLO LECs L_i^r, on the NNLO LECs C_i^r and the remainder as

$$m_a^{2(4)} = m_{a\,L}^{2(4)} + m_{a\,R}^{2(4)}\,, \qquad m_a^{2(6)} = m_{a\,L}^{2(6)} + m_{a\,C}^{2(6)} + m_{a\,R}^{2(6)}\,. \tag{72}$$

The expressions for these can be found in [21] and on [15]. Note that when combining these with results from other sources one should be sure to use a compatible LO and NLO.

Pion mass:
`double mpi4(physmass,Li)` returns $m_\pi^{2(4)}$
`double mpi4L(physmass,Li)` returns $m_{\pi\,L}^{2(4)}$
`double mpi4R(physmass,Li)` returns $m_{\pi\,R}^{2(4)}$

`double mpi6(physmass,Li,Ci)` returns $m_\pi^{2(6)}$
`double mpi6L(physmass,Li)` returns $m_{\pi L}^{2(6)}$
`double mpi6C(physmass,Ci)` returns $m_{\pi C}^{2(6)}$
`double mpi6R(physmass)` returns $m_{\pi R}^{2(6)}$

Kaon mass:
`double mk4(physmass,Li)` returns $m_K^{2(4)}$
`double mk4L(physmass,Li)` returns $m_{K L}^{2(4)}$
`double mk4R(physmass,Li)` returns $m_{K R}^{2(4)}$
`double mk6(physmass,Li,Ci)` returns $m_K^{2(6)}$
`double mk6L(physmass,Li)` returns $m_{K L}^{2(6)}$
`double mk6C(physmass,Ci)` returns $m_{K C}^{2(6)}$
`double mk6R(physmass)` returns $m_{K R}^{2(6)}$

Eta mass:
`double meta4(physmass,Li)` returns $m_\eta^{2(4)}$
`double meta4L(physmass,Li)` returns $m_{\eta L}^{2(4)}$
`double meta4R(physmass,Li)` returns $m_{\eta R}^{2(4)}$
`double meta6(physmass,Li,Ci)` returns $m_\eta^{2(6)}$
`double meta6L(physmass,Li)` returns $m_{\eta L}^{2(6)}$
`double meta6C(physmass,Ci)` returns $m_{\eta C}^{2(6)}$
`double meta6R(physmass)` returns $m_{\eta R}^{2(6)}$

The functions are defined in `massesdecayvev.h`, implemented in `massesdecayvev.cc` and examples of use are in `testmassdecayvev.cc`.

9.1.2 Decay constants

The decay constants of the pion, kaon and eta at two-loops in three flavour ChPT were obtained in [21]. The pion and eta decay constants were done earlier with a different subtraction scheme and a different way to perform the sunset integrals in [30].
The expressions for the decay constants for $a = \pi, K, \eta$ are given by

$$F_{a\,\mathrm{phys}} = F_0 \left(1 + F_a^{(4)} + F_a^{(6)}\right) . \tag{73}$$

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 [21], but note the description in the erratum of [31]. The expressions corrected for the error can be found in the website [15]. The normalization is such that $F_\pi \approx 92$ MeV.
The contributions themselves are divided into the parts depending on the NLO LECs L_i^r,

on the NNLO LECs C_i^r and the remainder as

$$F_a^{(4)} = F_{a\,L}^{(4)} + F_{a\,R}^{(4)}\,, \qquad F_a^{(6)} = F_{a\,L}^{(6)} + F_{a\,C}^{(6)} + F_{a\,R}^{(6)}\,. \tag{74}$$

For the η the decay constant has been defined with the octet axial-vector current.

Pion decay constant:
`double fpi4(physmass,Li)` returns $F_\pi^{(4)}$
`double fpi4L(physmass,Li)` returns $F_{\pi\,L}^{(4)}$
`double fpi4R(physmass,Li)` returns $F_{\pi\,R}^{(4)}$
`double fpi6(physmass,Li,Ci)` returns $F_\pi^{(6)}$
`double fpi6L(physmass,Li)` returns $F_{\pi\,L}^{(6)}$
`double fpi6C(physmass,Ci)` returns $F_{\pi\,C}^{(6)}$
`double fpi6R(physmass)` returns $F_{\pi\,R}^{(6)}$

Kaon decay constant:
`double fk4(physmass,Li)` returns $F_K^{(4)}$
`double fk4L(physmass,Li)` returns $F_{K\,L}^{(4)}$
`double fk4R(physmass,Li)` returns $F_{K\,R}^{(4)}$
`double fk6(physmass,Li,Ci)` returns $F_K^{(6)}$
`double fk6L(physmass,Li)` returns $F_{K\,L}^{(6)}$
`double fk6C(physmass,Ci)` returns $F_{K\,C}^{(6)}$
`double fk6R(physmass)` returns $F_{K\,R}^{(6)}$

Eta decay constant:
`double feta4(physmass,Li)` returns $F_\eta^{(4)}$
`double feta4L(physmass,Li)` returns $F_{\eta\,L}^{(4)}$
`double feta4R(physmass,Li)` returns $F_{\eta\,R}^{(4)}$
`double feta6(physmass,Li,Ci)` returns $F_\eta^{(6)}$
`double feta6L(physmass,Li)` returns $F_{\eta\,L}^{(6)}$
`double feta6C(physmass,Ci)` returns $F_{\eta\,C}^{(6)}$
`double feta6R(physmass)` returns $F_{\eta\,R}^{(6)}$

The functions are defined in `massesdecayvev.h`, implemented in `massesdecayvev.cc` and examples of use are in `testmassdecayvev.cc`.

9.1.3 `getfpimeta`

A problem that occurs in trying to compare to lattice QCD is that many of the routines are written in terms of the physical pion decay constant and physical 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 $m_\pi, m_K, m_\eta, F_\pi$.
With that input we can obtain an eta mass and pion decay constant with as input values the original `Liin`, `Ciin` and the `massin`. The formulas used are (72) and (74) up to order p^6 and p^4. The solution is obtained by iteration and stops when six digits of precision are reached. This method was used in [28] to obtain the consistent set of masses and decay constants used there.

```
physmass getfpimeta6(const double mpiin, const double mkin,
   const physmass massin,const Li Liin, const Ci Ciin)
```
returns a `physmass` containing `mpiin,mkin` and the calculated compatibe `meta,fpi` with the formulas including order p^6, i.e. to NNLO.
```
physmass getfpimeta4(const double mpiin, const double mkin,
   const physmass massin,const Li Liin)
```
returns a `physmass` containing `mpiin,mkin` and the calculated compatibe `meta,fpi` with the formulas including order p^4, i.e. to NLO.
The functions are defined in `getfpimeta.h`, implemented in `getfpimeta.cc` and examples of use are in `testgetfpimeta.cc`.

9.1.4 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 [31]. 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:

$$\langle 0|\overline{q}q|0\rangle_{a\,\mathrm{phys}} = -F_0^2 B_0 \left(1 + \langle 0|\overline{q}q|0\rangle_a^{(4)} + \langle 0|\overline{q}q|0\rangle_a^{(6)}\right) . \qquad (75)$$

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

$$\begin{aligned}\langle 0|\overline{q}q|0\rangle_a^{(4)} &= \langle 0|\overline{q}q|0\rangle_{a\,L}^{(4)} + \langle 0|\overline{q}q|0\rangle_{a\,R}^{(4)},\\ \langle 0|\overline{q}q|0\rangle_a^{(6)} &= \langle 0|\overline{q}q|0\rangle_{a\,L}^{(6)} + \langle 0|\overline{q}q|0\rangle_{a\,C}^{(6)} + \langle 0|\overline{q}q|0\rangle_{a\,R}^{(6)} .\end{aligned} \qquad (76)$$

These are defined for $q = u, s$.

$\langle 0|\overline{q}q|0\rangle_{u\,\mathrm{phys}}$:
`double qqup4(physmass,Li)` returns $\langle 0|\overline{q}q|0\rangle_u^{(4)}$
`double qqup4L(physmass,Li)` returns $\langle 0|\overline{q}q|0\rangle_{u\,L}^{(4)}$
`double qqup4R(physmass)` returns $\langle 0|\overline{q}q|0\rangle_{u\,R}^{(4)}$
`double qqup6(physmass,Li,Ci)` returns $\langle 0|\overline{q}q|0\rangle_u^{(6)}$
`double qqup6L(physmass,Li)` returns $\langle 0|\overline{q}q|0\rangle_{u\,L}^{(6)}$
`double qqup6C(physmass,Li)` returns $\langle 0|\overline{q}q|0\rangle_{u\,C}^{(6)}$
`double qqup6R(physmass)` returns $\langle 0|\overline{q}q|0\rangle_{u\,R}^{(6)}$

$\langle 0|\overline{q}q|0\rangle_{s\,\mathrm{phys}}$:
`double qqstrange4(physmass,Li)` returns $\langle 0|\overline{q}q|0\rangle_s^{(4)}$
`double qqstrange4L(physmass,Li)` returns $\langle 0|\overline{q}q|0\rangle_{s\,L}^{(4)}$
`double qqstrange4R(physmass)` returns $\langle 0|\overline{q}q|0\rangle_{s\,R}^{(4)}$
`double qqstrange6(physmass,Li,Ci)` returns $\langle 0|\overline{q}q|0\rangle_s^{(6)}$
`double qqstrange6L(physmass,Li)` returns $\langle 0|\overline{q}q|0\rangle_{s\,L}^{(6)}$
`double qqstrange6C(physmass,Li)` returns $\langle 0|\overline{q}q|0\rangle_{s\,C}^{(6)}$
`double qqstrange6R(physmass)` returns $\langle 0|\overline{q}q|0\rangle_{s\,R}^{(6)}$

The functions are defined in `massesdecayvev.h`, implemented in `massesdecayvev.cc` and examples of use are in `testmassesdecayvev.cc`

9.2 Masses, decay constants and vacuum-expectation-values: in lowest order

The exansions in this subsection are defined in terms of the lowest order masses, $m_{\pi\,0}, m_{K\,0}$, $m_\eta = \sqrt{(4m_{K\,0}^2 - m_{\pi\,0}^2)/3}$ and the lowest order, or chiral limit, pion decay constant F_0.

9.2.1 Masses: in lowest order

The masses of the pion, kaon and eta at two-loops in three flavour ChPT were evaluated in [21]. The pion and eta mass were done earlier with a different subtraction scheme and a different way to perform the sunset integrals in [30].
The expressions for the physical masses for $a = \pi, K, \eta$ are given by

$$m_{a\,\mathrm{phys}}^2 = m_{a\,0}^2 + m_a^{2(4)0} + m_a^{2(6)0}\,. \tag{77}$$

The superscripts indicate the order of the diagrams in p that each contribution comes from and the extra 0 that it is defined in terms of lowest-order quantities.
The lowest order masses are

$$m_{\pi\,0}^2 = 2B_0\hat{m}\,, \quad m_{K\,0}^2 = B_0\left(\hat{m} + m_s\right)\,, \quad m_{\eta\,0}^2 = \frac{2}{3}\left(\hat{m} + 2m_2\right)\,. \tag{78}$$

The higher order contributions are split in the parts depending on the NLO LECs L_i^r, on the NNLO LECs C_i^r and the remainder as

$$m_a^{2(4)0} = m_{a\,L}^{2(4)0} + m_{a\,R}^{2(4)0}\,, \qquad m_a^{2(6)0} = m_{a\,L}^{2(6)0} + m_{a\,C}^{2(6)0} + m_{a\,R}^{2(6)0}\,. \tag{79}$$

The expressions for were derived during the work for [21] and on [15]. Note that when combining these with results from other sources one should be sure to use a compatible LO and NLO.

Pion mass:
`double mpi4lo(lomass,Li)` returns $m_\pi^{2(4)0}$
`double mpi4Llo(lomass,Li)` returns $m_{\pi\,L}^{2(4)0}$
`double mpi4Rlo(lomass,Li)` returns $m_{\pi\,R}^{2(4)0}$
`double mpi6lo(lomass,Li,Ci)` returns $m_\pi^{2(6)0}$
`double mpi6Llo(lomass,Li)` returns $m_{\pi\,L}^{2(6)0}$
`double mpi6Clo(lomass,Ci)` returns $m_{\pi\,C}^{2(6)0}$
`double mpi6Rlo(lomass)` returns $m_{\pi\,R}^{2(6)0}$

Kaon mass:
`double mk4lo(lomass,Li)` returns $m_K^{2(4)0}$
`double mk4Llo(lomass,Li)` returns $m_{K\,L}^{2(4)0}$
`double mk4Rlo(lomass,Li)` returns $m_{K\,R}^{2(4)0}$
`double mk6lo(lomass,Li,Ci)` returns $m_K^{2(6)0}$
`double mk6Llo(lomass,Li)` returns $m_{K\,L}^{2(6)0}$
`double mk6Clo(lomass,Ci)` returns $m_{K\,C}^{2(6)0}$
`double mk6Rlo(lomass)` returns $m_{K\,R}^{2(6)0}$

Eta mass:
`double meta4lo(lomass,Li)` returns $m_\eta^{2(4)0}$
`double meta4Llo(lomass,Li)` returns $m_{\eta\,L}^{2(4)0}$
`double meta4Rlo(lomass,Li)` returns $m_{\eta\,R}^{2(4)0}$
`double meta6lo(lomass,Li,Ci)` returns $m_\eta^{2(6)0}$
`double meta6Llo(lomass,Li)` returns $m_{\eta\,L}^{2(6)0}$
`double meta6Clo(lomass,Ci)` returns $m_{\eta\,C}^{2(6)0}$
`double meta6Rlo(lomass)` returns $m_{\eta\,R}^{2(6)0}$

The functions are defined in `massesdecayvevlo.h`, implemented in `massesdecayvevlo.cc` and examples of use are in `testmassdecayvevlo.cc`.

9.2.2 Decay constants: in lowest order

The decay constants of the pion, kaon and eta at two-loops in three flavour ChPT were obtained in [21]. The pion and eta decay constants were done earlier with a different subtraction scheme and a different way to perform the sunset integrals in [30].
The expressions for the decay constants for $a = \pi, K, \eta$ are given by

$$F_{a\,\mathrm{phys}} = F_0 \left(1 + F_a^{(4)0} + F_a^{(6)0}\right) . \tag{80}$$

The superscripts indicate the order of the diagrams in p that each contribution comes from. The extra 0 indicates that the expansion is in terms of lowest-order quantities. F_0 denotes the decay constant in the three-flavour chiral limit. The expressions were originally derived during the work for [21] and can be found in the website [15]. The normalization is such that $F_\pi \approx 92$ MeV.
The contributions themselves are divided into the parts depending on the NLO LECs L_i^r, on the NNLO LECs C_i^r and the remainder as

$$F_a^{(4)0} = F_{aL}^{(4)0} + F_{aR}^{(4)0} , \qquad F_a^{(6)0} = F_{aL}^{(6)0} + F_{aC}^{(6)0} + F_{aR}^{(6)0} . \tag{81}$$

For the η the decay constant has been defined with the octet axial-vector current.

Pion decay constant:
`double fpi4lo(lomass,Li)` returns $F_\pi^{(4)0}$
`double fpi4Llo(lomass,Li)` returns $F_{\pi L}^{(4)0}$
`double fpi4Rlo(lomass,Li)` returns $F_{\pi R}^{(4)0}$
`double fpi6lo(lomass,Li,Ci)` returns $F_\pi^{(6)0}$
`double fpi6Llo(lomass,Li)` returns $F_{\pi L}^{(6)0}$
`double fpi6Clo(lomass,Ci)` returns $F_{\pi C}^{(6)0}$
`double fpi6Rlo(lomass)` returns $F_{\pi R}^{(6)0}$

Kaon decay constant:
`double fk4lo(lomass,Li)` returns $F_K^{(4)0}$
`double fk4Llo(lomass,Li)` returns $F_{KL}^{(4)0}$
`double fk4Rlo(lomass,Li)` returns $F_{KR}^{(4)0}$
`double fk6lo(lomass,Li,Ci)` returns $F_K^{(6)0}$
`double fk6Llo(lomass,Li)` returns $F_{KL}^{(6)0}$
`double fk6Clo(lomass,Ci)` returns $F_{KC}^{(6)0}$
`double fk6Rlo(lomass)` returns $F_{KR}^{(6)0}$

Eta decay constant:
`double feta4lo(lomass,Li)` returns $F_\eta^{(4)0}$
`double feta4Llo(lomass,Li)` returns $F_{\eta L}^{(4)0}$
`double feta4Rlo(lomass,Li)` returns $F_{\eta R}^{(4)0}$

`double feta6lo(lomass,Li,Ci)` returns $F_\eta^{(6)0}$
`double feta6Llo(lomass,Li)` returns $F_{\eta L}^{(6)0}$
`double feta6Clo(lomass,Ci)` returns $F_{\eta C}^{(6)0}$
`double feta6Rlo(lomass)` returns $F_{\eta R}^{(6)0}$

The functions are defined in `massesdecayvevlo.h`, implemented in `massesdecayvevlo.cc` and examples of use are in `testmassdecayvevlo.cc`.

9.2.3 Vacuum-expectation-values: in lowest order

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 [31]. 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:

$$\langle 0|\overline{q}q|0\rangle_{a\,\mathrm{phys}} = -F_0^2 B_0 \left(1 + \langle 0|\overline{q}q|0\rangle_a^{(4)0} + \langle 0|\overline{q}q|0\rangle_a^{(6)0}\right)\,. \tag{82}$$

The superscripts indicate the order of the diagrams in p that each contribution comes from. The extra 0 indicates that the expansion is defined in terms of lowest-order quantities. 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 massindependent, 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

$$\begin{aligned}
\langle 0|\overline{q}q|0\rangle_a^{(4)0} &= \langle 0|\overline{q}q|0\rangle_{aL}^{(4)0} + \langle 0|\overline{q}q|0\rangle_{aR}^{(4)0}\,,\\
\langle 0|\overline{q}q|0\rangle_a^{(6)0} &= \langle 0|\overline{q}q|0\rangle_{aL}^{(6)0} + \langle 0|\overline{q}q|0\rangle_{aC}^{(6)0} + \langle 0|\overline{q}q|0\rangle_{aR}^{(6)0}\,.
\end{aligned} \tag{83}$$

These are defined for $q = u, s$.

$\langle 0|\overline{q}q|0\rangle_{u\,\mathrm{phys}}$:
`double qqup4lo(lomass,Li)` returns $\langle 0|\overline{q}q|0\rangle_u^{(4)0}$
`double qqup4Llo(lomass,Li)` returns $\langle 0|\overline{q}q|0\rangle_{uL}^{(4)0}$
`double qqup4Rlo(lomass)` returns $\langle 0|\overline{q}q|0\rangle_{uR}^{(4)0}$
`double qqup6lo(lomass,Li,Ci)` returns $\langle 0|\overline{q}q|0\rangle_u^{(6)0}$
`double qqup6Llo(lomass,Li)` returns $\langle 0|\overline{q}q|0\rangle_{uL}^{(6)0}$

`double qqup6Clo(lomass,Li)` returns $\langle 0|\overline{q}q|0\rangle^{(6)0}_{u\,C}$
`double qqup6Rlo(lomass)` returns $\langle 0|\overline{q}q|0\rangle^{(6)0}_{u\,R}$

$\langle 0|\overline{q}q|0\rangle_{s\,\mathrm{phys}}$:
`double qqstrange4lo(lomass,Li)` returns $\langle 0|\overline{q}q|0\rangle^{(4)0}_{s}$
`double qqstrange4Llo(lomass,Li)` returns $\langle 0|\overline{q}q|0\rangle^{(4)0}_{s\,L}$
`double qqstrange4Rlo(lomass)` returns $\langle 0|\overline{q}q|0\rangle^{(4)0}_{s\,R}$
`double qqstrange6lo(lomass,Li,Ci)` returns $\langle 0|\overline{q}q|0\rangle^{(6)0}_{s}$
`double qqstrange6Llo(lomass,Li)` returns $\langle 0|\overline{q}q|0\rangle^{(6)0}_{s\,L}$
`double qqstrange6Clo(lomass,Li)` returns $\langle 0|\overline{q}q|0\rangle^{(6)0}_{s\,C}$
`double qqstrange6Rlo(lomass)` returns $\langle 0|\overline{q}q|0\rangle^{(6)0}_{s\,R}$

The functions are defined in `massesdecayvevlo.h`, implemented in `massesdecayvevlo.cc` and examples of use are in `testmassesdecayvevlo.cc`

9.3 Masses and decay constants at finite volume: in physical

The expressions treated in this section have been derived in [28]. 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, an overall factor at each order, and there is no 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. $\vec{p} = 0$.

9.3.1 Masses at finite volume: in physical

The finite volume corrections to the masses squared[5] are defined as the difference of the mass squared in finite volume and in infinite volume:

$$\begin{aligned}\Delta^V m_a^2 &= m_a^{2V} - m_a^{2\,V=\infty} = m_a^{2V(4)} + m_a^{2V(6)}\,.\\ m_a^{2V(6)} &= m_{a\,L}^{2V(6)} + m_{a\,R}^{2V(6)}\,.\end{aligned} \tag{84}$$

These definitions are for $a = \pi, K, \eta$.

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

Pion mass (theta function method):
`double mpi4Vt(const physmass massin,const double L)` returns $m_\pi^{2V(4)}$.
`double mpi6Vt(const physmass massin,const Li Liin,const double L)` returns $m_\pi^{2V(6)}$.
`double mpi6VLt(const physmass massin,const Li Liin,const double L)` returns $m_{\pi\,L}^{2V(6)}$.
`double mpi6VRt(const physmass massin,const double L)` returns $m_{\pi\,R}^{2V(6)}$.

Pion mass (Bessel function method):
`double mpi4Vb(const physmass massin,const double L)` returns $m_\pi^{2V(4)}$.
`double mpi6Vb(const physmass massin,const Li Liin,const double L)` returns $m_\pi^{2V(6)}$.
`double mpi6VLb(const physmass massin,const Li Liin,const double L)` returns $m_{\pi\,L}^{2V(6)}$.
`double mpi6VRb(const physmass massin,const double L)` returns $m_{\pi\,R}^{2V(6)}$.

Kaon mass (theta function method):
`double mk4Vt(const physmass massin,const double L)` returns $m_K^{2V(4)}$.
`double mk6Vt(const physmass massin,const Li Liin,const double L)` returns $m_K^{2V(6)}$.
`double mk6VLt(const physmass massin,const Li Liin,const double L)` returns $m_{K\,L}^{2V(6)}$.
`double mk6VRt(const physmass massin,const double L)` returns $m_{K\,R}^{2V(6)}$.

Kaon mass (Bessel function method):
`double mk4Vb(const physmass massin,const double L)` returns $m_K^{2V(4)}$.
`double mk6Vb(const physmass massin,const Li Liin,const double L)` returns $m_K^{2V(6)}$.
`double mk6VLb(const physmass massin,const Li Liin,const double L)` returns $m_{K\,L}^{2V(6)}$.
`double mk6VRb(const physmass massin,const double L)` returns $m_{K\,R}^{2V(6)}$.

Eta mass (theta function method):
`double meta4Vt(const physmass massin,const double L)` returns $m_\eta^{2V(4)}$.
`double meta6Vt(const physmass massin,const Li Liin,const double L)` returns $m_\eta^{2V(6)}$.
`double meta6VLt(const physmass massin,const Li Liin,const double L)` returns $m_{\eta\,L}^{2V(6)}$.
`double meta6VRt(const physmass massin,const double L)` returns $m_{\eta\,R}^{2V(6)}$.

Eta mass (Bessel function method):
`double meta4Vb(const physmass massin,const double L)` returns $m_\eta^{2V(4)}$.
`double meta6Vb(const physmass massin,const Li Liin,const double L)` returns $m_\eta^{2V(6)}$.
`double meta6VLb(const physmass massin,const Li Liin,const double L)` returns $m_{\eta\,L}^{2V(6)}$.
`double meta6VRb(const physmass massin,const double L)` returns $m_{\eta\,R}^{2V(6)}$.

All these are defined in `massdecayvevV.h` and implemented in `massdecayvevV.h`. Examples of use are in `testmassdecayvevV.cc`.

9.3.2 Decay constants at finite volume: in physical

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

$$\begin{aligned} \Delta^V F_a &= F_a^V - F_a^{V=\infty} = F_a^{V(4)} + F_a^{V(6)}\,. \\ F_a^{V(6)} &= F_{a\,L}^{V(6)} + F_{a\,R}^{V(6)}\,. \end{aligned} \tag{85}$$

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 the lowest order decay constant as in (73). The eta decay constant is defined with the octet axial current.

Pion decay constant (theta function method):
`double fpi4Vt(const physmass massin,const double L)` returns $F_\pi^{V(4)}$.
`double fpi6Vt(const physmass massin,const Li Liin,const double L)` returns $F_\pi^{V(6)}$.
`double fpi6VLt(const physmass massin,const Li Liin,const double L)` returns $F_{\pi\,L}^{V(6)}$.
`double fpi6VRt(const physmass massin,const double L)` returns $F_{\pi\,R}^{V(6)}$.

Pion decay constant (Bessel function method):
`double fpi4Vb(const physmass massin,const double L)` returns $F_\pi^{V(4)}$.
`double fpi6Vb(const physmass massin,const Li Liin,const double L)` returns $F_\pi^{V(6)}$.
`double fpi6VLb(const physmass massin,const Li Liin,const double L)` returns $F_{\pi\,L}^{V(6)}$.
`double fpi6VRb(const physmass massin,const double L)` returns $F_{\pi\,R}^{V(6)}$.

Kaon decay constant (theta function method):
`double fk4Vt(const physmass massin,const double L)` returns $F_K^{V(4)}$.
`double fk6Vt(const physmass massin,const Li Liin,const double L)` returns $F_K^{V(6)}$.
`double fk6VLt(const physmass massin,const Li Liin,const double L)` returns $F_{K\,L}^{V(6)}$.
`double fk6VRt(const physmass massin,const double L)` returns $F_{K\,R}^{V(6)}$.

Kaon decay constant (Bessel function method):
`double fk4Vb(const physmass massin,const double L)` returns $F_K^{V(4)}$.
`double fk6Vb(const physmass massin,const Li Liin,const double L)` returns $F_K^{V(6)}$.
`double fk6VLb(const physmass massin,const Li Liin,const double L)` returns $F_{K\,L}^{V(6)}$.
`double fk6VRb(const physmass massin,const double L)` returns $F_{K\,R}^{V(6)}$.

Eta decay constant (theta function method):
`double feta4Vt(const physmass massin,const double L)` returns $F_\eta^{V(4)}$.
`double feta6Vt(const physmass massin,const Li Liin,const double L)` returns $F_\eta^{V(6)}$.
`double feta6VLt(const physmass massin,const Li Liin,const double L)` returns $F_{\eta\,L}^{V(6)}$.
`double feta6VRt(const physmass massin,const double L)` returns $F_{\eta\,R}^{V(6)}$.

Eta decay constant (Bessel function method):
`double feta4Vb(const physmass massin,const double L)` returns $F_\eta^{V(4)}$.
`double feta6Vb(const physmass massin,const Li Liin,const double L)` returns $F_\eta^{V(6)}$.
`double feta6VLb(const physmass massin,const Li Liin,const double L)` returns $F_{\eta L}^{V(6)}$.
`double feta6VRb(const physmass massin,const double L)` returns $F_{\eta R}^{V(6)}$.

All these are defined in `massdecayvevV.h` and implemented in `massdecayvevV.h`. Examples of use are in `testmassdecayvevV.cc`.

9.4 Masses, decay constants and vacuum expectation values at finite volume: in lowest order

The expressions treated in this section have been derived in [28]. 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, an overall factor at each order, and there is no 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. $\vec{p} = 0$.

9.4.1 Masses at finite volume: in lowest order

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:

$$\begin{aligned}\Delta^V m_a^2 &= m_a^{2V} - m_a^{2\,V=\infty} = m_a^{2V(4)0} + m_a^{2V(6)0}\,.\\ m_a^{2V(6)0} &= m_{a\,L}^{2V(6)0} + m_{a\,R}^{2V(6)0}\,.\end{aligned} \tag{86}$$

These definitions are for $a = \pi, K, \eta$.

Pion mass (theta function method):
`double mpi4loVt(const lomass massin,const double L)` returns $m_\pi^{2V(4)0}$.
`double mpi6loVt(const lomass massin,const Li Liin,const double L)` returns $m_\pi^{2V(6)0}$.
`double mpi6LloVt(const lomass massin,const Li Liin,const double L)` returns $m_{\pi L}^{2V(6)0}$.

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

`double mpi6RloVt(const lomass massin,const double L)` returns $m^{2V(6)0}_{\pi R}$.

Pion mass (Bessel function method):
`double mpi4loVb(const lomass massin,const double L)` returns $m^{2V(4)0}_{\pi}$.
`double mpi6loVb(const lomass massin,const Li Liin,const double L)` returns $m^{2V(6)0}_{\pi}$.
`double mpi6LloVb(const lomass massin,const Li Liin,const double L)` returns $m^{2V(6)0}_{\pi L}$.
`double mpi6RloVb(const lomass massin,const double L)` returns $m^{2V(6)0}_{\pi R}$.

Kaon mass (theta function method):
`double mk4loVt(const lomass massin,const double L)` returns $m^{2V(4)0}_{K}$.
`double mk6loVt(const lomass massin,const Li Liin,const double L)` returns $m^{2V(6)0}_{K}$.
`double mk6LloVt(const lomass massin,const Li Liin,const double L)` returns $m^{2V(6)0}_{K L}$.
`double mk6RloVt(const lomass massin,const double L)` returns $m^{2V(6)0}_{K R}$.

Kaon mass (Bessel function method):
`double mk4loVb(const lomass massin,const double L)` returns $m^{2V(4)0}_{K}$.
`double mk6loVb(const lomass massin,const Li Liin,const double L)` returns $m^{2V(6)0}_{K}$.
`double mk6LloVb(const lomass massin,const Li Liin,const double L)` returns $m^{2V(6)0}_{K L}$.
`double mk6RloVb(const lomass massin,const double L)` returns $m^{2V(6)}_{K R}$.

Eta mass (theta function method):
`double meta4loVt(const lomass massin,const double L)` returns $m^{2V(4)0}_{\eta}$.
`double meta6loVt(const lomass massin,const Li Liin,const double L)` returns $m^{2V(6)0}_{\eta}$.
`double meta6LloVt(const lomass massin,const Li Liin,const double L)` returns $m^{2V(6)0}_{\eta L}$.
`double meta6RloVt(const lomass massin,const double L)` returns $m^{2V(6)0}_{\eta R}$.

Eta mass (Bessel function method):
`double meta4loVb(const lomass massin,const double L)` returns $m^{2V(4)0}_{\eta}$.
`double meta6loVb(const lomass massin,const Li Liin,const double L)` returns $m^{2V(6)0}_{\eta}$.
`double meta6LloVb(const lomass massin,const Li Liin,const double L)` returns $m^{2V(6)0}_{\eta L}$.
`double meta6RloVb(const lomass massin,const double L)` returns $m^{2V(6)0}_{\eta R}$.

All these are defined in `massdecayvevloV.h` and implemented in `massdecayvevloV.h`. Examples of use are in `testmassdecayvevloV.cc`.

9.4.2 Decay constants at finite volume: in lowest order

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

$$\Delta^V F_a \;=\; F_a^V - F_a^{V=\infty} = F_0\left(F_a^{V(4)0} + F_a^{V(6)0}\right)\,.$$

$$F_a^{V(6)0} = F_{a\,L}^{V(6)0} + F_{a\,R}^{V(6)0}\,. \qquad (87)$$

Note that this is a different normalization compared to the expressions in terms of physical masses and the physical F_π. This was done to have the same normalization as the partially quenched results. These definitions are for $a = \pi, K, \eta$. The correction is defined to the value of the decay constant divided by the the lowest order decay constant as in (73). The eta decay constant is defined with the octet axial current.

Pion decay constant (theta function method):
`double fpi4loVt(const lomass massin,const double L)` returns $F_\pi^{V(4)0}$.
`double fpi6loVt(const lomass massin,const Li Liin,const double L)` returns $F_\pi^{V(6)0}$.
`double fpi6LloVt(const lomass massin,const Li Liin,const double L)` returns $F_{\pi\,L}^{V(6)0}$.
`double fpi6RloVt(const lomass massin,const double L)` returns $F_{\pi\,R}^{V(6)0}$.

Pion decay constant (Bessel function method):
`double fpi4loVb(const lomass massin,const double L)` returns $F_\pi^{V(4)0}$.
`double fpi6loVb(const lomass massin,const Li Liin,const double L)` returns $F_\pi^{V(6)0}$.
`double fpi6LloVb(const lomass massin,const Li Liin,const double L)` returns $F_{\pi\,L}^{V(6)0}$.
`double fpi6RloVb(const lomass massin,const double L)` returns $F_{\pi\,R}^{V(6)0}$.

Kaon decay constant (theta function method):
`double fk4loVt(const lomass massin,const double L)` returns $F_K^{V(4)0}$.
`double fk6loVt(const lomass massin,const Li Liin,const double L)` returns $F_K^{V(6)0}$.
`double fk6LloVt(const lomass massin,const Li Liin,const double L)` returns $F_{K\,L}^{V(6)0}$.
`double fk6RloVt(const lomass massin,const double L)` returns $F_{K\,R}^{V(6)0}$.

Kaon decay constant (Bessel function method):
`double fk4loVb(const lomass massin,const double L)` returns $F_K^{V(4)0}$.
`double fk6loVb(const lomass massin,const Li Liin,const double L)` returns $F_K^{V(6)0}$.
`double fk6LloVb(const lomass massin,const Li Liin,const double L)` returns $F_{K\,L}^{V(6)0}$.
`double fk6RloVb(const lomass massin,const double L)` returns $F_{K\,R}^{V(6)0}$.

Eta decay constant (theta function method):
`double feta4loVt(const lomass massin,const double L)` returns $F_\eta^{V(4)0}$.
`double feta6loVt(const lomass massin,const Li Liin,const double L)` returns $F_\eta^{V(6)0}$.
`double feta6LloVt(const lomass massin,const Li Liin,const double L)` returns $F_{\eta\,L}^{V(6)0}$.
`double feta6RloVt(const lomass massin,const double L)` returns $F_{\eta\,R}^{V(6)0}$.

Eta decay constant (Bessel function method):
`double feta4loVb(const lomass massin,const double L)` returns $F_\eta^{V(4)0}$.
`double feta6loVb(const lomass massin,const Li Liin,const double L)` returns $F_\eta^{V(6)0}$.

`double feta6LloVb(const lomass massin,const Li Liin,const double L)` returns $F^{V(6)0}_{\eta\,L}$.
`double feta6RloVb(const lomass massin,const double L)` returns $F^{V(6)0}_{\eta\,R}$.

All these are defined in `massdecayvevloV.h` and implemented in `massdecayvevloV.h`. Examples of use are in `testmassdecayvevloV.cc`.

9.4.3 Vacuum-expectation-values at finite volume: in lowest order

The finite volume 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 [29]. The expression for the up and down quark vev are identical since we are in the isospin limit. The finite volume correction is defined as the difference between the infinite and finite volume value.
We write the expressions in a form analoguous to the decay constant treatment:

$$\Delta^V\langle 0|\overline{q}q|0\rangle \equiv \langle 0|\overline{q}q|0\rangle^V_{a\,\mathrm{phys}} - \langle 0|\overline{q}q|0\rangle^{V=\infty}_{a\,\mathrm{phys}} = -F_0^2B_0\left(\langle 0|\overline{q}q|0\rangle^{V(4)0}_a + \langle 0|\overline{q}q|0\rangle^{V(6)0}_a\right)\,. \quad (88)$$

The superscripts indicate the order of the diagrams in p that each contribution comes from. The extra 0 indicates that the expansion is defined in terms of lowest-order quantities. The lowest order values are $-F_0^2B_0$.
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_0m_q is independent of it. 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 and the remainder as

$$\langle 0|\overline{q}q|0\rangle^{(6)0}_a \;=\; \langle 0|\overline{q}q|0\rangle^{(6)0}_{a\,L} + \langle 0|\overline{q}q|0\rangle^{(6)0}_{a\,R}\,. \quad (89)$$

These are defined for $q = u, s$.

The last letter `x` is `b` when the finite volume integrals are calculated using the Bessel finction method is `t` when the theta function method is used. Do not forget to set the precision wanted for the finite volume integrals. `L` is the length of the three spatial directions.
$\Delta^V\langle 0|\overline{q}q|0\rangle_{u\,\mathrm{phys}}$:
`double qqup4loVx(lomass,L)` returns $\langle 0|\overline{q}q|0\rangle^{V(4)0}_u$
`double qqup6loVx(lomass,Li,L)` returns $\langle 0|\overline{q}q|0\rangle^{V(6)0}_u$
`double qqup6LloVx(lomass,Li,L)` returns $\langle 0|\overline{q}q|0\rangle^{V(6)0}_{u\,L}$
`double qqup6RloVx(lomass,L)` returns $\langle 0|\overline{q}q|0\rangle^{V(6)0}_{u\,R}$

$\Delta^V \langle 0|\overline{q}q|0\rangle_{s\,\mathrm{phys}}$:

`double qqstrange4loVx(lomass,L)` returns $\langle 0|\overline{q}q|0\rangle_s^{V(4)0}$
`double qqstrange6loVx(lomass,Li,L)` returns $\langle 0|\overline{q}q|0\rangle_s^{V(6)0}$
`double qqstrange6LloVx(lomass,Li,L)` returns $\langle 0|\overline{q}q|0\rangle_{s\,L}^{V(6)0}$
`double qqstrange6RloVx(lomass,L)` returns $\langle 0|\overline{q}q|0\rangle_{s\,R}^{V(6)0}$

The functions are defined in `massesdecayvevloV.h`, implemented in `massesdecayvevloV.cc` and examples of use are in `testmassesdecayvevloV.cc`

9.5 Masses, decay constants and vacuum expectation values at finite volume with twisted boundary conditions: in lowest order

Twisted boundary conditions for a given field are defined by (43) or

$$\Phi(x^i + L) = e^{i\theta^i}\Phi(x^i), . \tag{90}$$

Chiral perturbation theory with twisted boundary conditions was discussed in [36]. The formulas in this section give the finite volume correction in terms of lowest order masses and decay constant for the case $m_u = m_d \neq m_s$. These formulas contain the case of a moving frame as the cases with twist-angles zero.

The subroutines expect as input the lowest order mass of π and K, F_0 and μ via a `lomass`, the length of the spatial volume, fourvectors containing in the spatial components the twist-angles for up, down and strange quark and a fourvector containing the momentum which should be compatible (not checked!) with the twist angles and length of the finite volume.
As an example, the spatial momentum components of a K^0 consisting of a down and antistrange quark must be of the form

$$p^i = \frac{1}{L}\left(2\pi n^i + \theta_d^i - \theta_s^i\right) , \tag{91}$$

with n^i positive or negative integers.

9.5.1 Masses

The masses were derived in [35], see there for earlier work. The result gives the full finite volume correction to one-loop order $\Delta^V m^2(4)0$. The infinite volume correction, Sect. 9.2.1, must be added separately. So up to one-loop order the masses are

$$m_a^2 = m_a^2(2)0 + m_a^{2(4)0} + \Delta^V m_a^{2(4)0} . \tag{92}$$

Note that the mass of the anti-particle is the same as the particle *with the spatial momentum of the opposite sign*, we have not added functions for π^-, K^- and $\overline{K^0}$.

`const lomass mass(mp0, mk0, F0, mu)`: lowest order pion and kaon mass, pion decay constant and subtractions scale
`const double L`: The length of the spatial volume, assumed to be the same in all three directions.
`const fourvector p`: The four-momentum of the particle. Only the spatial components are used and must be consistent with the twist angles and the size of the volumes.
`const fourvector thetu,thetad,thetas`: The spatial components contain the twist angles for the up, down and strange quark in the three spatial directions.

$\pi^+, \pi^0, K^+, K^0, \eta$:
`double mpipp4TV(mass, L, p, thetau, thetad, thetas)` returns $\Delta^V m^{2(4)0}_{\pi^+}$
`double mpiop4TV(mass, L, p, thetau, thetad, thetas)` returns $\Delta^V m^{2(4)0}_{\pi^0}$
`double mkpp4TV(mass, L, p, thetau, thetad, thetas)` returns $\Delta^V m^{2(4)0}_{K^+}$
`double mkop4TV(mass, L, p, thetau, thetad, thetas)` returns $\Delta^V m^{2(4)0}_{K^0}$
`double metap4TV(mass, L, p, thetau, thetad, thetas)` returns $\Delta^V m^{2(4)0}_{\eta}$

Defined in `massdecayvevTV.h`, implemented in `massdecayvevTV.cc` and examples of use in `testmassdecayvevTV.cc`.

9.6 Vector form-factors: in lowest order

9.6.1 Electromagnetic form-factors

The electromagnetic form-factors are defined by

$$\langle M(p')|\frac{2}{3}\overline{u}\gamma_\mu u(0) - \frac{1}{3}\overline{d}\gamma_\mu d(0) - \frac{1}{3}\overline{s}\gamma_\mu s(0)|M(p)\rangle = \left(p'_\mu + p_\mu\right) F_V^M(t)\,, \tag{93}$$

with $t = (p' - p)^2$. The electromagnetic form-factor vanishes identically for $M = \pi^0, \eta$ because of charge conjugation.
These were first calculated at one-loop in ChPT [37], the two-loop expressions were obtained in [22].
The form-factors are (up to order p^4) given as

$$\begin{aligned} F_V^M(t) =& F_V^{M(2)0} + F_V^{M(4)0}(t)\,,\\ F_V^{M(4)0}(t) =& F_{VL}^{M(4)0}(t) + F_{VR}^{M(4)0}(t),. \end{aligned} \tag{94}$$

The p^4 part is split in an L_i^r dependent part and the remainder. The lowest order is either one or zero and $F_{VL}^{M(4)0}(t)$ is similarly simple

$$\begin{aligned} F_V^{\pi^+(2)0} =& F_V^{K^+(2)0} = 1\,, \qquad F_V^{K^0(2)0} = 0\,.\\ F_{VL}^{\pi^+(4)0}(t) =& F_{VL}^{K^+(4)0}(t) = 2tL_9^r\,, \qquad F_{VL}^{K^0(4)0} = 0\,. \end{aligned} \tag{95}$$

The form-factors for $\pi^-, K^-, \overline{K}^0$ are given by charge conjugation and are the opposite sign of those for π^+, K^+, K^0. The p^4 then depends on whether you have pulled out that overall sign or not in your conventions.

9.6.1.1 Functions

Inputs are: `const double t, const lomass mass, const Li Liin`.
π^+:
`dcomplex fvpipp4lo(t,mass,Liin)` returns $F_V^{\pi^+(4)0}(t)$
`double fvpipp4Llo(t,mass,Liin)` returns $F_{VL}^{\pi^+(4)0}(t)$
`dcomplex fvpipp4Rlo(t,mass)` returns $F_{VR}^{\pi^+(4)0}(t)$

K^+:
`dcomplex fvkpp4lo(t,mass,Liin)` returns $F_V^{K^+(4)0}(t)$
`double fvkpp4Llo(t,mass,Liin)` returns $F_{VL}^{K^+(4)0}(t)$
`dcomplex fvkpp4Rlo(t,mass)` returns $F_{VR}^{K^+(4)0}(t)$

K^0:
`dcomplex fvkop4lo(t,mass,Liin)` returns $F_V^{K^0(4)0}(t)$
`double fvkop4Llo(t,mass,Liin)` returns $F_{VL}^{K^0(4)0}(t) = 0$
`dcomplex fvkop4Rlo(t,mass)` returns $F_{VR}^{K^0(4)0}(t)$

Defined in `vectorformlo.h`, implemented in `vectorformlo.cc` and examples of use in `testvectorformlo.cc`

9.6.2 $K \to \pi$ and $K_{\ell 3}$ form-factors

The $K \to \pi$ and $K_{\ell 3}$ form-factors are defined by

$$\langle \pi^-(p')|\bar{s}\gamma_\mu u(0)|K^0(p)\rangle = \left(p'_\mu + p_\mu\right) f_+^{K\pi}(t) + \left(p'_\mu - p_\mu\right) f_-^{K\pi}(t)\,, \tag{96}$$

with $t = (p' - p)^2$. The form-factors for the decay $K^+ \to \pi^0$ are the same up to an overall factor of $1/\sqrt{2}$.
These were first calculated at one-loop in ChPT [37], the two-loop expressions were obtained in [38].
The form-factors are (up to order p^4) given as

$$\begin{aligned} f_\pm^{K\pi}(t) =& f_\pm^{K\pi(2)0} + f_\pm^{K\pi(4)0}(t)\,, \\ f_\pm^{K\pi(4)0}(t) =& f_{\pm L}^{K\pi(4)0}(t) + f_{\pm R}^{K\pi(4)0}(t),\,. \end{aligned} \tag{97}$$

The p^4 part is split in an L_i^r dependent part and the remainder. The lowest order is one for $f_+^{K\pi}$ zero for $f_-^{K\pi}$. Note that the definition of the scalar form-factor

$$f_0^{K\pi}(t) = f_+^{K\pi}(t) + \frac{t}{m_K^2 - m_\pi^2} f_-^{K\pi}(t) \tag{98}$$

involves the physical masses. To one-loop order this is not relevant but it is when working to p^6 or two-loops.
The form-factors for $K^-, \overline{K}^0$ are given by charge conjugation and are the opposite sign of those for K^+, K^0. The p^4 then depends on whether you have pulled out that overall sign or not in your conventions.

9.6.2.1 Functions

Inputs are: `const double t, const lomass mass, const Li Liin`.
`dcomplex fvpkpip4lo(t,mass,Liin)` returns $f_+^{K\pi(4)0}(t)$
`double fvpkpip4Llo(t,mass,Liin)` returns $f_{+L}^{K\pi(4)0}(t)$
`dcomplex fvpkpip4Rlo(t,mass)` returns $f_{+R}^{K\pi(4)0}(t)$

`dcomplex fvmkpip4lo(t,mass,Liin)` returns $f_-^{K\pi(4)0}(t)$
`double fvmkpip4Llo(t,mass,Liin)` returns $f_{-L}^{K\pi(4)0}(t)$
`dcomplex fvmkpip4Rlo(t,mass)` returns $f_{-R}^{K\pi(4)0}(t)$

Defined in `vectorformlo.h`, implemented in `vectorformlo.cc` and examples of use in `testvectorformlo.cc`

10 Three flavour partially quenched results

This section contains the routines used for the partially quenched results with three sea quark flavours of [24, 26, 27]. The formulas used are analytically equivalent to those in the published papers, but are longer and avoid some of the 0/0 problems that can appear. The finite volume expressions were derived in [32].
Do not forget to set the precision for the needed sunset integrals with `setprecisionquenchedsunsetintegral` and the finite volume equivalents.
The interface is always defined with the n_F flavour NLO and NNLO LECs `Linf` and `Ki` with $n_F = 3$. The routines also expect a `quarkmassnf` with precisely the number of quark masses needed for each case.
The reason why the quarkmasses or alternatively lowest-order meson masses are used is that in these cases there are very many physical masses compared to the number of quark masses. There would thus have been a very large ambiguity in expressing the results in physical masses.
The inputs used are $Bm_1 = m_{11}^2/2$, $Bm_2 = m_{22}^2/2$, $Bm_3 = m_{33}^2/2$, $Bm_4 = m_{44}^2/2$, $Bm_5 = m_{55}^2/2$, $Bm_6 = m_{66}^2/2$.
We give the cases for equal or different valence quark mass ,cases `v1` or `v2`, and one, two or three different sea quark masses, cases `s1`, `s2`, `s3` for always three sea flavours, case `nf3`. For the one sea mass case we have $Bm_4 = Bm_5 = Bm_6$ and for the two sea mass case $Bm_4 = Bm_5$.

The quark masses The masses are labelled starting with `m` and the decay constants starting with `f`. `f0` is F_0 the three flavour chiral limit decay constant and `mu` is the subtraction scale μ.
For the finite volume cases the names have an additional `V` and a `b` or `t` dependending on whether the Bessel function or the theta function method is used for the finite volume integrals. `L` is the spatial extent of the finite directions.

10.1 Masses

The expansion are defined similar to (77) via

$$m^2_{a\,\mathrm{phys}} = m^2_{a\,0} + m_a^{2(4)0} + m_a^{2(6)0}\,. \tag{99}$$

The masses are for the off-diagonal or charged meson with two different valence quarks but the masses can be equal or different.

$$m_a^{2(4)0} = m^{2(4)0}_{a\,L} + m^{2(4)0}_{a\,R}\,,\qquad m_a^{2(6)0} = m^{2(6)0}_{a\,L} + m^{2(6)0}_{a\,K} + m^{2(6)0}_{a\,R}\,. \tag{100}$$

One valence mass, one sea mass:
`mass=quarkmassnf({Bm1,Bm4},f0,mu,2)`
`double mv1s1nf3p4(quarkmassnf mass, Linf Liin)` returns $m_a^{2(4)0}$
`double mv1s1nf3p4L(quarkmassnf mass, Linf Liin)` returns $m^{2(4)0}_{a\,L}$
`double mv1s1nf3p4R(quarkmassnf mass)` returns $m^{2(4)0}_{a\,R}$
`double mv1s1nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $m_a^{2(6)0}$
`double mv1s1nf3p6L(quarkmassnf mass, Linf Liin)` returns $m^{2(6)0}_{a\,L}$
`double mv1s1nf3p6K(quarkmassnf mass, Ki Kiin)` returns $m^{2(6)0}_{a\,K}$
`double mv1s1nf3p6R(quarkmassnf mass)` returns $m^{2(6)0}_{a\,R}$

Two valence mass, one sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4},f0,mu,3)`
`double mv2s1nf3p4(quarkmassnf mass, Linf Liin)` returns $m_a^{2(4)0}$
`double mv2s1nf3p4L(quarkmassnf mass, Linf Liin)` returns $m^{2(4)0}_{a\,L}$
`double mv2s1nf3p4R(quarkmassnf mass)` returns $m^{2(4)0}_{a\,R}$
`double mv2s1nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $m_a^{2(6)0}$
`double mv2s1nf3p6L(quarkmassnf mass, Linf Liin)` returns $m^{2(6)0}_{a\,L}$
`double mv2s1nf3p6K(quarkmassnf mass, Ki Kiin)` returns $m^{2(6)0}_{a\,K}$
`double mv2s1nf3p6R(quarkmassnf mass)` returns $m^{2(6)0}_{a\,R}$

One valence mass, two sea mass:
`mass=quarkmassnf({Bm1,Bm4,Bm6},f0,mu,3)`

`double mv1s1nf3p4(quarkmassnf mass, Linf Liin)` returns $m_a^{2(4)0}$
`double mv1s1nf3p4L(quarkmassnf mass, Linf Liin)` returns $m_{a\,L}^{2(4)0}$
`double mv1s1nf3p4R(quarkmassnf mass)` returns $m_{a\,R}^{2(4)0}$
`double mv1s1nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $m_a^{2(6)0}$
`double mv1s1nf3p6L(quarkmassnf mass, Linf Liin)` returns $m_{a\,L}^{2(6)0}$
`double mv1s1nf3p6K(quarkmassnf mass, Ki Kiin)` returns $m_{a\,K}^{2(6)0}$
`double mv1s1nf3p6R(quarkmassnf mass)` returns $m_{a\,R}^{2(6)0}$

Two valence mass, two sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4,Bm6},f0,mu,4)`
`double mv2s2nf3p4(quarkmassnf mass, Linf Liin)` returns $m_a^{2(4)0}$
`double mv2s2nf3p4L(quarkmassnf mass, Linf Liin)` returns $m_{a\,L}^{2(4)0}$
`double mv2s2nf3p4R(quarkmassnf mass)` returns $m_{a\,R}^{2(4)0}$
`double mv2s2nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $m_a^{2(6)0}$
`double mv2s2nf3p6L(quarkmassnf mass, Linf Liin)` returns $m_{a\,L}^{2(6)0}$
`double mv2s2nf3p6K(quarkmassnf mass, Ki Kiin)` returns $m_{a\,K}^{2(6)0}$
`double mv2s2nf3p6R(quarkmassnf mass)` returns $m_{a\,R}^{2(6)0}$

One valence mass, three sea mass:
`mass=quarkmassnf({Bm1,Bm4,Bm5,Bm6},f0,mu,4)`
`double mv1s3nf3p4(quarkmassnf mass, Linf Liin)` returns $m_a^{2(4)0}$
`double mv1s3nf3p4L(quarkmassnf mass, Linf Liin)` returns $m_{a\,L}^{2(4)0}$
`double mv1s3nf3p4R(quarkmassnf mass)` returns $m_{a\,R}^{2(4)0}$
`double mv1s3nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $m_a^{2(6)0}$
`double mv1s3nf3p6L(quarkmassnf mass, Linf Liin)` returns $m_{a\,L}^{2(6)0}$
`double mv1s3nf3p6K(quarkmassnf mass, Ki Kiin)` returns $m_{a\,K}^{2(6)0}$
`double mv1s3nf3p6R(quarkmassnf mass)` returns $m_{a\,R}^{2(6)0}$

Two valence mass, three sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4,Bm5,Bm6},f0,mu,5)`
`double mv2s3nf3p4(quarkmassnf mass, Linf Liin)` returns $m_a^{2(4)0}$
`double mv2s3nf3p4L(quarkmassnf mass, Linf Liin)` returns $m_{a\,L}^{2(4)0}$
`double mv2s3nf3p4R(quarkmassnf mass)` returns $m_{a\,R}^{2(4)0}$
`double mv2s3nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $m_a^{2(6)0}$
`double mv2s3nf3p6L(quarkmassnf mass, Linf Liin)` returns $m_{a\,L}^{2(6)0}$
`double mv2s3nf3p6K(quarkmassnf mass, Ki Kiin)` returns $m_{a\,K}^{2(6)0}$
`double mv2s3nf3p6R(quarkmassnf mass)` returns $m_{a\,R}^{2(6)0}$

Defined in `massdecayvevPQ.h`, implemented in `massdecayvevPQ.cc` and examples of use in `testmassdecayvevPQ.cc`.

10.2 Decay constants

The expansion are defined similar to (81) via

$$F_{a\,\mathrm{phys}} = F_0 \left(1 + F_a^{(4)0} + F_a^{(6)0}\right) . \tag{101}$$

The decay constants are for the off-diagonal or charged meson with two different valence quarks but the masses can be equal or different. The normalization corresponds to the pion decay constant $F_\pi \approx 92$ MeV.

$$F_a^{(4)0} = F_{a\,L}^{(4)0} + F_{a\,R}^{(4)0}\,, \qquad F_a^{(6)0} = F_{a\,L}^{(6)0} + F_{a\,K}^{(6)0} + F_{a\,R}^{(6)0}\,. \tag{102}$$

One valence mass, one sea mass:
`mass=quarkmassnf({Bm1,Bm4},f0,mu,2)`
`double fv1s1nf3p4(quarkmassnf mass, Linf Liin)` returns $F_a^{(4)0}$
`double fv1s1nf3p4L(quarkmassnf mass, Linf Liin)` returns $F_{a\,L}^{(4)0}$
`double fv1s1nf3p4R(quarkmassnf mass)` returns $F_{a\,R}^{(4)0}$
`double fv1s1nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $F_a^{(6)0}$
`double fv1s1nf3p6L(quarkmassnf mass, Linf Liin)` returns $F_{a\,L}^{(6)0}$
`double fv1s1nf3p6K(quarkmassnf mass, Ki Kiin)` returns $F_{a\,K}^{(6)0}$
`double fv1s1nf3p6R(quarkmassnf mass)` returns $F_{a\,R}^{(6)0}$

Two valence mass, one sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4},f0,mu,3)`
`double fv2s1nf3p4(quarkmassnf mass, Linf Liin)` returns $F_a^{(4)0}$
`double fv2s1nf3p4L(quarkmassnf mass, Linf Liin)` returns $F_{a\,L}^{(4)0}$
`double fv2s1nf3p4R(quarkmassnf mass)` returns $F_{a\,R}^{(4)0}$
`double fv2s1nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $F_a^{(6)0}$
`double fv2s1nf3p6L(quarkmassnf mass, Linf Liin)` returns $F_{a\,L}^{(6)0}$
`double fv2s1nf3p6K(quarkmassnf mass, Ki Kiin)` returns $F_{a\,K}^{(6)0}$
`double fv2s1nf3p6R(quarkmassnf mass)` returns $F_{a\,R}^{(6)0}$

One valence mass, two sea mass:
`mass=quarkmassnf({Bm1,Bm4,Bm6},f0,mu,3)`
`double fv1s1nf3p4(quarkmassnf mass, Linf Liin)` returns $F_a^{(4)0}$
`double fv1s1nf3p4L(quarkmassnf mass, Linf Liin)` returns $F_{a\,L}^{(4)0}$
`double fv1s1nf3p4R(quarkmassnf mass)` returns $F_{a\,R}^{(4)0}$
`double fv1s1nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $F_a^{(6)0}$
`double fv1s1nf3p6L(quarkmassnf mass, Linf Liin)` returns $F_{a\,L}^{(6)0}$
`double fv1s1nf3p6K(quarkmassnf mass, Ki Kiin)` returns $F_{a\,K}^{(6)0}$

`double fv1s1nf3p6R(quarkmassnf mass)` returns $F^{(6)0}_{a\,R}$

Two valence mass, two sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4,Bm6},f0,mu,4)`
`double fv2s2nf3p4(quarkmassnf mass, Linf Liin)` returns $F^{(4)0}_{a}$
`double fv2s2nf3p4L(quarkmassnf mass, Linf Liin)` returns $F^{(4)0}_{a\,L}$
`double fv2s2nf3p4R(quarkmassnf mass)` returns $F^{(4)0}_{a\,R}$
`double fv2s2nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $F^{(6)0}_{a}$
`double fv2s2nf3p6L(quarkmassnf mass, Linf Liin)` returns $F^{(6)0}_{a\,L}$
`double fv2s2nf3p6K(quarkmassnf mass, Ki Kiin)` returns $F^{(6)0}_{a\,K}$
`double fv2s2nf3p6R(quarkmassnf mass)` returns $F^{(6)0}_{a\,R}$

One valence mass, three sea mass:
`mass=quarkmassnf({Bm1,Bm4,Bm5,Bm6},f0,mu,4)`
`double fv1s3nf3p4(quarkmassnf mass, Linf Liin)` returns $F^{(4)0}_{a}$
`double fv1s3nf3p4L(quarkmassnf mass, Linf Liin)` returns $F^{(4)0}_{a\,L}$
`double fv1s3nf3p4R(quarkmassnf mass)` returns $F^{(4)0}_{a\,R}$
`double fv1s3nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $F^{(6)0}_{a}$
`double fv1s3nf3p6L(quarkmassnf mass, Linf Liin)` returns $F^{(6)0}_{a\,L}$
`double fv1s3nf3p6K(quarkmassnf mass, Ki Kiin)` returns $F^{(6)0}_{a\,K}$
`double fv1s3nf3p6R(quarkmassnf mass)` returns $F^{(6)0}_{a\,R}$

Two valence mass, three sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4,Bm5,Bm6},f0,mu,5)`
`double fv2s3nf3p4(quarkmassnf mass, Linf Liin)` returns $F^{(4)0}_{a}$
`double fv2s3nf3p4L(quarkmassnf mass, Linf Liin)` returns $F^{(4)0}_{a\,L}$
`double fv2s3nf3p4R(quarkmassnf mass)` returns $F^{(4)0}_{a\,R}$
`double fv2s3nf3p6(quarkmassnf mass, Linf Liin, Ki Kiin)` returns $F^{(6)0}_{a}$
`double fv2s3nf3p6L(quarkmassnf mass, Linf Liin)` returns $F^{(6)0}_{a\,L}$
`double fv2s3nf3p6K(quarkmassnf mass, Ki Kiin)` returns $F^{(6)0}_{a\,K}$
`double fv2s3nf3p6R(quarkmassnf mass)` returns $F^{(6)0}_{a\,R}$

Defined in `massdecayvevPQ.h`, implemented in `massdecayvevPQ.cc` and examples of use in `testmassdecayvevPQ.cc`.

10.3 Masses at finite volume

The expressions treated in this section have been derived in [32]. It contains the routines for the finite volume corrections for the masses of the off-diagonal or charged mesons in partially quenched ChPT with three sea quark flavours.

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, an overall factor at each order, and there is no 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. $\vec{p} = 0$.
The expansion are defined similar to (86) via

$$\begin{aligned}\Delta^V m_a^2 &= m_a^{2V} - m_a^{2\,V=\infty} = m_a^{2V(4)0} + m_a^{2V(6)0}\,.\\ m_a^{2V(6)0} &= m_{a\,L}^{2V(6)0} + m_{a\,R}^{2V(6)0}\,.\end{aligned} \tag{103}$$

The masses are for the off-diagonal or charged meson with two different valence quarks but the masses can be equal or different.
`x` should be `b` or `t` depening on whether you want to use the finite volume integrals using bessel functions or theta functions.

One valence mass, one sea mass:
`mass=quarkmassnf({Bm1,Bm4},f0,mu,2)`
`double mv1s1nf3p4Vx(quarkmassnf mass, double L)` returns $m_a^{2V(4)0}$
`double mv1s1nf3p6Vx(quarkmassnf mass, Linf Liin,double L)` returns $m_a^{2V(6)0}$
`double mv1s1nf3p6LVx(quarkmassnf mass, Linf Liin, double L)` returns $m_{a\,L}^{2V(6)0}$
`double mv1s1nf3p6RVx(quarkmassnf mass, double L)` returns $m_{a\,R}^{2V(6)0}$

Two valence mass, one sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4},f0,mu,3)`
`double mv2s1nf3p4Vx(quarkmassnf mass, double L)` returns $m_a^{2V(4)0}$
`double mv2s1nf3p6Vx(quarkmassnf mass, Linf Liin, double L)` returns $m_a^{2V(6)0}$
`double mv2s1nf3p6LVx(quarkmassnf mass, Linf Liin, double L)` returns $m_{a\,L}^{2V(6)0}$
`double mv2s1nf3p6RVx(quarkmassnf mass, double L)` returns $m_{a\,R}^{2V(6)0}$

One valence mass, two sea mass:
`mass=quarkmassnf({Bm1,Bm4,Bm6},f0,mu,3)`
`double mv1s1nf3p4Vx(quarkmassnf mass, double L)` returns $m_a^{2V(4)0}$
`double mv1s1nf3p6Vx(quarkmassnf mass, Linf Liin, double L)` returns $m_a^{2V(6)0}$
`double mv1s1nf3p6LVx(quarkmassnf mass, Linf Liin, double L)` returns $m_{a\,L}^{2V(6)0}$

`double mv1s1nf3p6RVx(quarkmassnf mass, double L)` returns $m^{2V(6)0}_{a\,R}$

Two valence mass, two sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4,Bm6},f0,mu,4)`
`double mv2s2nf3p4Vx(quarkmassnf mass, double L)` returns $m^{2V(4)0}_{a}$
`double mv2s2nf3p6Vx(quarkmassnf mass, Linf Liin, double L)` returns $m^{2V(6)0}_{a}$
`double mv2s2nf3p6LVx(quarkmassnf mass, Linf Liin, double L)` returns $m^{2V(6)0}_{a\,L}$
`double mv2s2nf3p6RVx(quarkmassnf mass, double L)` returns $m^{2V(6)0}_{a\,R}$

One valence mass, three sea mass:
`mass=quarkmassnf({Bm1,Bm4,Bm5,Bm6},f0,mu,4)`
`double mv1s3nf3p4Vx(quarkmassnf mass, double L)` returns $m^{2V(4)0}_{a}$
`double mv1s3nf3p6Vx(quarkmassnf mass, Linf Liin, double L)` returns $m^{2V(6)0}_{a}$
`double mv1s3nf3p6LVx(quarkmassnf mass, Linf Liin, double L)` returns $m^{2V(6)0}_{a\,L}$
`double mv1s3nf3p6RVx(quarkmassnf mass, double L)` returns $m^{2V(6)0}_{a\,R}$

Two valence mass, three sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4,Bm5,Bm6},f0,mu,5)`
`double mv2s3nf3p4Vx(quarkmassnf mass, double L)` returns $m^{2V(4)0}_{a}$
`double mv2s3nf3p6Vx(quarkmassnf mass, Linf Liin, double L)` returns $m^{2V(6)0}_{a}$
`double mv2s3nf3p6LVx(quarkmassnf mass, Linf Liin, double L)` returns $m^{2V(6)0}_{a\,L}$
`double mv2s3nf3p6RVx(quarkmassnf mass, double L)` returns $m^{2V(6)0}_{a\,R}$

Defined in `massdecayvevPQV.h`, implemented in `massdecayvevPQV.cc` and examples of use in `testmassdecayvevPQV.cc`.

10.4 Decay constants at finite volume

The expressions treated in this section have been derived in [32]. It contains the routines for the finite volume corrections for the decay constants of the off-diagonal or charged mesons in partially quenched ChPT with three sea quark flavours.
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, an overall factor at each order, and there is no dependence on the NLO LECs L^r_i.

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. $\vec{p} = 0$.
The expansion are defined similar to (87) via

$$\begin{aligned}\Delta^V F_a &= F_a^V - F_a^{V=\infty} = F_0\left(F_a^{V(4)0} + F_a^{V(6)0}\right) .\\ F_a^{V(6)0} &= F_{a\,L}^{V(6)0} + F_{a\,R}^{V(6)0} .\end{aligned} \tag{104}$$

The decay constants are for the off-diagonal or charged meson with two different valence quarks but the masses can be equal or different.
`x` should be `b` or `t` depening on whether you want to use the finite volume integrals using bessel functions or theta functions.

One valence mass, one sea mass:
`mass=quarkmassnf({Bm1,Bm4},f0,mu,2)`
`double fv1s1nf3p4Vx(quarkmassnf mass, double L)` returns $F_a^{V(4)0}$
`double fv1s1nf3p6Vx(quarkmassnf mass, Linf Liin, double L)` returns $F_a^{V(6)0}$
`double fv1s1nf3p6LVx(quarkmassnf mass, Linf Liin, double L)` returns $F_{a\,L}^{V(6)0}$
`double fv1s1nf3p6RVx(quarkmassnf mass, double L)` returns $F_{a\,R}^{V(6)0}$

Two valence mass, one sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4},f0,mu,3)`
`double fv2s1nf3p4Vx(quarkmassnf mass, double L)` returns $F_a^{V(4)0}$
`double fv2s1nf3p6Vx(quarkmassnf mass, Linf Liin, double L)` returns $F_a^{V(6)0}$
`double fv2s1nf3p6LVx(quarkmassnf mass, Linf Liin, double L)` returns $F_{a\,L}^{V(6)0}$
`double fv2s1nf3p6RVx(quarkmassnf mass, double L)` returns $F_{a\,R}^{V(6)0}$

One valence mass, two sea mass:
`mass=quarkmassnf({Bm1,Bm4,Bm6},f0,mu,3)`
`double fv1s1nf3p4Vx(quarkmassnf mass, double L)` returns $F_a^{V(4)0}$
`double fv1s1nf3p6Vx(quarkmassnf mass, Linf Liin, double L)` returns $F_a^{V(6)0}$
`double fv1s1nf3p6LVx(quarkmassnf mass, Linf Liin, double L)` returns $F_{a\,L}^{V(6)0}$
`double fv1s1nf3p6RVx(quarkmassnf mass, double L)` returns $F_{a\,R}^{V(6)0}$

Two valence mass, two sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4,Bm6},f0,mu,4)`
`double fv2s2nf3p4(quarkmassnf mass, double L)` returns $F_a^{V(4)0}$
`double fv2s2nf3p6(quarkmassnf mass, Linf Liin, double L)` returns $F_a^{V(6)0}$
`double fv2s2nf3p6L(quarkmassnf mass, Linf Liin, double L)` returns $F_{a\,L}^{V(6)0}$
`double fv2s2nf3p6R(quarkmassnf mass, double L)` returns $F_{a\,R}^{V(6)0}$

One valence mass, three sea mass:
`mass=quarkmassnf({Bm1,Bm4,Bm5,Bm6},f0,mu,4)`
`double fv1s3nf3p4Vx(quarkmassnf mass, double L)` returns $F_a^{V(4)0}$
`double fv1s3nf3p6Vx(quarkmassnf mass, Linf Liin, double L)` returns $F_a^{V(6)0}$
`double fv1s3nf3p6LVx(quarkmassnf mass, Linf Liin, double L)` returns $F_{a\,L}^{V(6)0}$
`double fv1s3nf3p6RVx(quarkmassnf mass, double L)` returns $F_{a\,R}^{V(6)0}$

Two valence mass, three sea mass:
`mass=quarkmassnf({Bm1,Bm3,Bm4,Bm5,Bm6},f0,mu,5)`
`double fv2s3nf3p4Vx(quarkmassnf mass, double L)` returns $F_a^{V(4)0}$
`double fv2s3nf3p6Vx(quarkmassnf mass, Linf Liin, double L)` returns $F_a^{V(6)0}$
`double fv2s3nf3p6LVx(quarkmassnf mass, Linf Liin, double L)` returns $F_{a\,L}^{V(6)0}$
`double fv2s3nf3p6RVx(quarkmassnf mass, double L)` returns $F_{a\,R}^{V(6)0}$

Defined in `massdecayvevPQV.h`, implemented in `massdecayvevPQV.cc` and examples of use in `testmassdecayvevPQV.cc`.

10.5 Vector form-factors

10.5.1 $K \to \pi$ or $K_{\ell 3}$ form-factors

We define the form-factors connecting an off-diagonal or charged meson consisting of valence quark content $q_{\text{val2}}\bar{q}_{\text{val3}}$ to another off-diagonal or charged meson with valence quark content $q_{\text{val2}}\bar{q}_{\text{val1}}$ similar to (96).

$$\langle \pi_{q_{\text{val2}}\bar{q}_{\text{val1}}}(p')|\bar{q}_{\text{val3}}\gamma_\mu q_{\text{val1}}(0)|\pi_{q_{\text{val2}}\bar{q}_{\text{val3}}}(p)\rangle = \left(p'_\mu + p_\mu\right) f_+^{PQ}(t) + \left(p'_\mu - p_\mu\right) f_-^{PQ}(t)\,, \quad (105)$$

with $t = (p'-p)^2$. With 1 the up valence, 2 the down valence and three the strange valence quark this corresponds to the form-factors for $K^0 \to \pi^-$. Note that all three valence quarks are different but can have the same masses.
The formfactor f_+^{PQ} to order p^4 is

$$\begin{aligned} f_+^{PQ}(t) =& 1 + f_+^{PQ(4)}(t)\,, \\ f_+^{PQ(4)}(t) =& f_{L+}^{PQ(4)}(t) + f_{R+}^{PQ(4)}(t) \end{aligned} \quad (106)$$

The p^4 is split in the L_i^r dependent part and the remainder.
The results for the f_+^{PQ} to order p^4 were derived in [39].

10.5.2 Functions

The L_i^r dependent part is to order p^4 independent of the masses and is $2L_9^r t/F_0^2$.

The inputs are all defined as `const`.

All mass cases:
`mass=quarkmassnf({Bm1,...,Bm6},f0,mu,2)`
`double fvpnf3p4LV(double t, quarkmassnf mass, Linf Liin)` returns $f^{PQ(4)}_{L+}(t)$

2 valence and 2 sea quark masses, note $Bm_2 = Bm_1$, $Bm_5 = Bm_4$:
`mass=quarkmassnf({Bm1,Bm3,Bm4,Bm6},f0,mu,2)`
`double fvpv2s2nf3p4R(double t, quarkmassnf mass)` returns $f^{PQ(4)}_{R+}(t)$

2 valence and 3 sea quark masses, note $Bm_2 = Bm_1$:
`mass=quarkmassnf({Bm1,Bm3,Bm4,Bm5,Bm6},f0,mu,2)`
`double fvpv2s3nf3p4R(double t, quarkmassnf mass)` returns $f^{PQ(4)}_{R+}(t)$

3 valence and 2 sea quark masses, note $Bm_5 = Bm_4$:
`mass=quarkmassnf({Bm1,Bm2,Bm3,Bm4,Bm6},f0,mu,2)`
`double fvpv3s2nf3p4R(double t, quarkmassnf mass)` returns $f^{PQ(4)}_{R+}(t)$

3 valence and 3 sea quark masses:
`mass=quarkmassnf({Bm1,Bm2,Bm3,Bm4,Bm5,Bm6},f0,mu,2)`
`double fvpv3s3nf3p4R(double t, quarkmassnf mass)` returns $f^{PQ(4)}_{R+}(t)$

Note that since $f^{PQ(4)}_{+}$ does not depend on the mass of the spectator valence quark mass (Bm_2) the three valence mass case simply calls the two valence mass case with $Bm2$ removed.

Defined in `vectorformPQ.h`, implemented in `testvectorformPQ.h` and examples of use in `testvectorformPQ.cc`.

11 QCD like theories for N_F flavours

There are other symmetry breaking patterns possible then the one used in two and three-flavour ChPT. With N_F Dirac fermions in a complex, real or pseudoreal representation the global symmetries are $SU(N_F) \times SU(N_F)$, $SU(2N_F)$ and $SU(2N_F)$. This is described in [33] and references therein. The symmetry breaking pattern in these case is down to the subgroups the diagonal $SU(N_F)$, $SO(2N_F)$ and $Sp(2N_F)$ respectyively. An additional case is n_F Majorana fermions in a real representation. In this case the global symmetry group is $SU(n_F)$ which is expected to be spontaneously broken to $SO(n_F)$, see [34] and references therein. The formulas for the two cases with real fermions are identical with $n_F = 2N_F$, the reason is that the two cases are related by a $U(2N_F)$ rotation as explained in [34].
The number of flavours N_F in this section refers to the symmetry breaking patterns $SU(N_F) \times SU(N_F) \to SU(N_F)$, $SU(N_F) \to SO(N_F)$ and $SU(2N_F) \to Sp(2N_F)$. The

number of flavours for the real representation case thus counts the number of Majorana fermions and is in the case twice the N_F used in [33]. The extension to finite volume and partially quenched was done in [34].
For all cases we only treat the simplest mass case. It means that for the unquenched case we have a single mass given by $B_0 m_1$ or the lowest order meson mass is $m^2_{LO} = 2B_0 m_1$. For the partially quenched case the sea quark mass is given by $B_0 m_4$ or the lowest order mass for sea quark mesons is $m^2_{LO} = 2B_0 m_4$.
The standard C++ conversions allow the routines to be called using a `lomassnf` instead of a `quarkmassnf`.

11.1 Mass, decay constant and vacuum-expectation-value: in lowest order

The functions in this section return the corrections to the mass-squared in terms of the quarkmassnf structure. It is assumed that all the quarks have the same mass. The label `XXX=SUN,SON,SPN` refers to the three possible patterns of symmetry breaking, $SU(N_F) \times SU(N_F) \to SU(N_F)$, $SU(N_F) \to SO(N_F)$ and $SU(2N_F) \to Sp(2N_F)$ and `nf`$=N_F$ as defined in this way.
The mass is defined as in [33] as

$$\begin{aligned} M^2_{\text{phys}} =& M^2 + M^{(4)2} + M^{(6)2}\,, \\ M^{(4)2} =& M^{(4)2}_L + M^{(4)2}_R\,, \\ M^{(6)2} =& M^{(6)2}_K + M^{(6)2}_L + M^{(6)2}_R\,, \end{aligned} \tag{107}$$

The expressions are defined in terms of lowest order masses and decay constant.
`mass=quarkmassnf({Bm1},f0,mu,1)`
`double mnfXXXp4(int nf, quarkmassnf mass, Liinf Liin)` returns $M^{(4)2}$
`double mnfXXXp4L(int nf, quarkmassnf mass, Liinf Liin)` returns $M^{(4)2}_L$
`double mnfXXXp4R(int nf, quarkmassnf mass)` returns $M^{(4)2}_R$
`double mnfXXX6(int nf, quarkmassnf mass, Liinf Liin, Ki Kiin)` returns $M^{(6)2}$
`double mnfXXXp6K(int nf, quarkmassnf mass, Ki Kiin)` returns $M^{(6)2}_K$
`double mnfXXXp6L(int nf, quarkmassnf mass, Liinf Liin)` returns $M^{(6)2}_L$
`double mnfXXXp6R(int nf, quarkmassnf mass)` returns $M^{(6)2}_R$

The functions `mnfXXXp6K` for `XXX=SON,SPN` simply return 0. The Lagrangians for these cases have not been classified at NNLO or order p^6. The interface used here is also with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L^r_{11} is always included even if it plays no role. In addition the running of the LECs is not correct for `XXX=SON,SPN`.

The decay constant is defined as in [33] as

$$\begin{aligned}F_{\rm phys} =&F_0\left(1+F^{(4)2}+F^{(6)2}\right)\,,\\ F^{(4)} =&F^{(4)}_L+F^{(4)}_R\,,\\ F^{(6)} =&F^{(6)}_K+F^{(6)}_L+M^{(6)}_R\,.\end{aligned}\tag{108}$$

The expressions are defined in terms of lowest order masses and decay constant.
`mass=quarkmassnf({Bm1},f0,mu,1)`
`double fnfXXXp4(int nf, quarkmassnf mass, Liinf Liin)` returns $F^{(4)}$
`double fnfXXXp4L(int nf, quarkmassnf mass, Liinf Liin)` returns $F^{(4)}_L$
`double fnfXXXp4R(int nf, quarkmassnf mass)` returns $F^{(4)}_R$
`double fnfXXX6(int nf, quarkmassnf mass, Liinf Liin, Ki Kiin)` returns $F^{(6)}$
`double fnfXXXp6K(int nf, quarkmassnf mass, Ki Kiin)` returns $F^{(6)}_K$
`double fnfXXXp6L(int nf, quarkmassnf mass, Liinf Liin)` returns $F^{(6)}_L$
`double fnfXXXp6R(int nf, quarkmassnf mass)` returns $F^{(6)}_R$

The functions `fnfXXXp6K` for `XXX=SON,SPN` simply return 0. The Lagrangians for these cases have not been classified at NNLO or order p^6. The interface used here is also with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L^r_{11} is always included even if it plays no role. In addition the running of the LECs is not correct for `XXX=SON,SPN`.

The vacuum expectation value is defined for a single quark similar to [33] as

$$\begin{aligned}\langle\bar qq\rangle_{\rm phys} =&-B_0F_0^2\left(1+\langle\bar qq\rangle^{(4)}+\langle\bar qq\rangle^{(6)}\right)\,,\\ \langle\bar qq\rangle^{(4)} =&\langle\bar qq\rangle^{(4)}_L+\langle\bar qq\rangle^{(4)}_R\,,\\ \langle\bar qq\rangle^{(6)} =&\langle\bar qq\rangle^{(6)}_K+\langle\bar qq\rangle^{(6)}_L+\langle\bar qq\rangle^{(6)}_R\,.\end{aligned}\tag{109}$$

The expressions are defined in terms of lowest order masses and decay constant.
`mass=quarkmassnf({Bm1},f0,mu,1)`
`double qnfXXXp4(int nf, quarkmassnf mass, Liinf Liin)` returns $\langle\bar qq\rangle^{(4)}$
`double qnfXXXp4L(int nf, quarkmassnf mass, Liinf Liin)` returns $\langle\bar qq\rangle^{(4)}_L$
`double qnfXXXp4R(int nf, quarkmassnf mass)` returns $\langle\bar qq\rangle^{(4)}_R$
`double qnfXXX6(int nf, quarkmassnf mass, Liinf Liin, Ki Kiin)` returns $\langle\bar qq\rangle^{(6)}$
`double qnfXXXp6K(int nf, quarkmassnf mass, Ki Kiin)` returns $\langle\bar qq\rangle^{(6)}_K$
`double qnfXXXp6L(int nf, quarkmassnf mass, Liinf Liin)` returns $\langle\bar qq\rangle^{(6)}_L$
`double qnfXXXp6R(int nf, quarkmassnf mass)` returns $\langle\bar qq\rangle^{(6)}_R$

The functions `qnfXXXp6K` for `XXX=SON,SPN` simply return 0. The Lagrangians for these cases have not been classified at NNLO or order p^6. The interface used here is also with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L^r_{11} is always included even if it plays no role. In addition the running of the LECs is not correct for `XXX=SON,SPN`.

Defined in `massdecayvevnf.h`, implemented in `massdecayvevnf.cc` and examples of use in `testmassdecayvevnf.cc`.

11.2 Mass, decay constant and vacuum-expectation-value at finite volume: in lowest order

The functions in this section return the finite volume corrections to the mass-squared in terms of the quarkmassnf structure. It is assumed that all the quarks have the same mass. The label `XXX=SUN,SON,SPN` refers to the three possible patterns of symmetry breaking, $SU(N_F)\times SU(N_F)\to SU(N_F)$, $SU(N_F)\to SO(N_F)$ and $SU(2N_F)\to Sp(2N_F)$ and `nf`$=N_F$ as defined in this way.
The last letter `x` is `b` for the finite volume integrals evaluated using the Bessel function method or `t` when they are evaluated using the theta function method.

The finite volume correction to the mass is defined as in [34] as

$$\begin{aligned}\Delta^V M^2_{\text{phys}} \equiv& M^{2\,V}_{\text{phys}} - M^{2\,V=\infty}_{\text{phys}} = M^{V(4)2} + M^{V(6)2}\,,\\ M^{V(6)2} =& M_L^{V(6)2} + M_R^{V(6)2}\,,\end{aligned}\tag{110}$$

The expressions are defined in terms of lowest order masses and decay constant. `L` is the size of the spatial directions.
`mass=quarkmassnf({Bm1},f0,mu,1)`
`double mnfXXXp4Vx(int nf, quarkmassnf mass, double L)` returns $M^{V(4)2}$
`double mnfXXX6Vx(int nf, quarkmassnf mass, Liinf Liin, double L)` returns $M^{V(6)2}$
`double mnfXXXp6LVx(int nf, quarkmassnf mass, Liinf Liin, double L)` returns $M_L^{V(6)2}$
`double mnfXXXp6RVx(int nf, quarkmassnf mass, double L)` returns $M_R^{V(6)2}$

The interface used here is with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L^r_{11} is always included even if it plays no role. In addition the running of the LECs is not correct for `XXX=SON,SPN`.

The finite volume correction to the decay constant is defined as in [34] as

$$\begin{aligned}\Delta^V F_{\text{phys}} \equiv& F^V_{\text{phys}} - F^{V=\infty}_{\text{phys}} = F_0\left(F^{V(4)2} + F^{V(6)2}\right)\,,\\ F^{V(6)} =& F_L^{V(6)} + M_R^{V(6)}\,.\end{aligned}\tag{111}$$

The expressions are defined in terms of lowest order masses and decay constant.

`mass=quarkmassnf({Bm1},f0,mu,1)`
`double fnfXXXp4Vx(int nf, quarkmassnf mass, double L)` returns $F^{V(4)}$
`double fnfXXX6Vx(int nf, quarkmassnf mass, Liinf Liin, double L)` returns $F^{V(6)}$
`double fnfXXXp6LVx(int nf, quarkmassnf mass, Liinf Liin, double L)` returns $F_L^{V(6)}$
`double fnfXXXp6RVx(int nf, quarkmassnf mass, double L)` returns $F_R^{V(6)}$

The interface used here is with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L_{11}^r is always included even if it plays no role. In addition the running of the LECs is not correct for `XXX=SON,SPN`.

The vacuum expectation value is defined for a single quark similar to [34] as

$$\begin{aligned}\Delta^V\langle\bar{q}q\rangle_{\text{phys}} \equiv&\langle\bar{q}q\rangle^V_{\text{phys}} - \langle\bar{q}q\rangle^{V=\infty}_{\text{phys}} = -B_0F_0^2\left(\langle\bar{q}q\rangle^{V(4)} + \langle\bar{q}q\rangle^{V(6)}\right)\,,\\ \langle\bar{q}q\rangle^{(6)} =&\langle\bar{q}q\rangle^{(6)}_L + \langle\bar{q}q\rangle^{(6)}_R\,.\end{aligned}\tag{112}$$

The expressions are defined in terms of lowest order masses and decay constant.
`mass=quarkmassnf({Bm1},f0,mu,1)`
`double qnfXXXp4Vx(int nf, quarkmassnf mass, double L)` returns $\langle\bar{q}q\rangle^{V(4)}$
`double qnfXXX6Vx(int nf, quarkmassnf mass, Liinf Liin, Ki Kiin)` returns $\langle\bar{q}q\rangle^{V(6)}$
`double qnfXXXp6LVx(int nf, quarkmassnf mass, Liinf Liin)` returns $\langle\bar{q}q\rangle_L^{V(6)}$
`double qnfXXXp6RVx(int nf, quarkmassnf mass)` returns $\langle\bar{q}q\rangle_R^{V(6)}$

The interface used here is with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L_{11}^r is always included even if it plays no role. In addition the running of the LECs is not correct for `XXX=SON,SPN`.

Defined in `massdecayvevnfV.h`, implemented in `massdecayvevnfV.cc` and examples of use in `testmassdecayvevnf.cc`.

11.3 Partially quenched mass, decay constant and vacuum-expectation-value: in lowest order

The functions in this section return the corrections to the mass-squared in terms of the quarkmassnf structure. It is assumed that all the valence quarks have the same mass and all the sea quarks have the same mass but different from the valence quarks.
The label `XXX=SUN,SON,SPN` refers to the three possible patterns of symmetry breaking, $SU(N_F)\times SU(N_F)\to SU(N_F)$, $SU(N_F)\to SO(N_F)$ and $SU(2N_F)\to Sp(2N_F)$ and `nf`$=N_F$ as defined in this way.
The mass is defined as in [34] as

$$M^2_{\text{phys}} = M^2 + M^{(4)2} + M^{(6)2}\,,$$

$$
\begin{aligned}
M^{(4)2} =& M_L^{(4)2} + M_R^{(4)2}\,,\\
M^{(6)2} =& M_K^{(6)2} + M_L^{(6)2} + M_R^{(6)2}\,,
\end{aligned}
\qquad (113)
$$

The expressions are defined in terms of lowest order masses and decay constant.
`mass=quarkmassnf({Bm1,Bm4},f0,mu,2)`
`double mnfXXXp4PQ(int nf, quarkmassnf mass, Liinf Liin)` returns $M^{(4)2}$
`double mnfXXXp4LPQ(int nf, quarkmassnf mass, Liinf Liin)` returns $M_L^{(4)2}$
`double mnfXXXp4RPQ(int nf, quarkmassnf mass)` returns $M_R^{(4)2}$
`double mnfXXX6PQ(int nf, quarkmassnf mass, Liinf Liin, Ki Kiin)` returns $M^{(6)2}$
`double mnfXXXp6KPQ(int nf, quarkmassnf mass, Ki Kiin)` returns $M_K^{(6)2}$
`double mnfXXXp6LPQ(int nf, quarkmassnf mass, Liinf Liin)` returns $M_L^{(6)2}$
`double mnfXXXp6RPQ(int nf, quarkmassnf mass)` returns $M_R^{(6)2}$

The functions `mnfXXXp6KPQ` for `XXX=SON,SPN` simply return 0. The Lagrangians for these cases have not been classified at NNLO or order p^6. The interface used here is also with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L_{11}^r is always included even if it plays no role. In addition the running of the LECs is not correct for `XXX=SON,SPN`.

The decay constant is defined as in [34] as

$$
\begin{aligned}
F_{\text{phys}} =& F_0\left(1 + F^{(4)2} + F^{(6)2}\right)\,,\\
F^{(4)} =& F_L^{(4)} + F_R^{(4)}\,,\\
F^{(6)} =& F_K^{(6)} + F_L^{(6)} + M_R^{(6)}\,.
\end{aligned}
\qquad (114)
$$

The expressions are defined in terms of lowest order masses and decay constant.
`mass=quarkmassnf({Bm1,Bm4},f0,mu,2)`
`double fnfXXXp4PQ(int nf, quarkmassnf mass, Liinf Liin)` returns $F^{(4)}$
`double fnfXXXp4LPQ(int nf, quarkmassnf mass, Liinf Liin)` returns $F_L^{(4)}$
`double fnfXXXp4RPQ(int nf, quarkmassnf mass)` returns $F_R^{(4)}$
`double fnfXXX6PQ(int nf, quarkmassnf mass, Liinf Liin, Ki Kiin)` returns $F^{(6)}$
`double fnfXXXp6KPQ(int nf, quarkmassnf mass, Ki Kiin)` returns $F_K^{(6)}$
`double fnfXXXp6LPQ(int nf, quarkmassnf mass, Liinf Liin)` returns $F_L^{(6)}$
`double fnfXXXp6RPQ(int nf, quarkmassnf mass)` returns $F_R^{(6)}$

The functions `fnfXXXp6KPQ` for `XXX=SON,SPN` simply return 0. The Lagrangians for these cases have not been classified at NNLO or order p^6. The interface used here is also with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L_{11}^r is always included even if it plays no role. In addition the running of the LECs is not correct for

`XXX=SON,SPN`.

The vacuum expectation value is defined for a single quark similar to [34] as

$$\begin{aligned}\langle\bar qq\rangle_{\text{phys}} =& -B_0F_0^2\left(1+\langle\bar qq\rangle^{(4)}+\langle\bar qq\rangle^{(6)}\right)\,,\\ \langle\bar qq\rangle^{(4)} =&\langle\bar qq\rangle^{(4)}_L+\langle\bar qq\rangle^{(4)}_R\,,\\ \langle\bar qq\rangle^{(6)} =&\langle\bar qq\rangle^{(6)}_K+\langle\bar qq\rangle^{(6)}_L+\langle\bar qq\rangle^{(6)}_R\,.\end{aligned} \tag{115}$$

The expressions are defined in terms of lowest order masses and decay constant.
`mass=quarkmassnf({Bm1,Bm4},f0,mu,2)`
`double qnfXXXp4PQ(int nf, quarkmassnf mass, Liinf Liin)` returns $\langle\bar qq\rangle^{(4)}$
`double qnfXXXp4LPQ(int nf, quarkmassnf mass, Liinf Liin)` returns $\langle\bar qq\rangle^{(4)}_L$
`double qnfXXXp4RPQ(int nf, quarkmassnf mass)` returns $\langle\bar qq\rangle^{(4)}_R$
`double qnfXXX6PQ(int nf, quarkmassnf mass, Liinf Liin, Ki Kiin)` returns $\langle\bar qq\rangle^{(6)}$
`double qnfXXXp6KPQ(int nf, quarkmassnf mass, Ki Kiin)` returns $\langle\bar qq\rangle^{(6)}_K$
`double qnfXXXp6LPQ(int nf, quarkmassnf mass, Liinf Liin)` returns $\langle\bar qq\rangle^{(6)}_L$
`double qnfXXXp6RPQ(int nf, quarkmassnf mass)` returns $\langle\bar qq\rangle^{(6)}_R$

The functions `qnfXXXp6KPQ` for `XXX=SON,SPN` simply return 0. The Lagrangians for these cases have not been classified at NNLO or order p^6. The interface used here is also with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L^r_{11} is always included even if it plays no role. In addition the running of the LECs is not correct for `XXX=SON,SPN`.

Defined in `massdecayvevnfPQ.h`, implemented in `massdecayvevnfPQ.cc` and examples of use in `testmassdecayvevnf.cc`.

11.4 Partially quenched mass, decay constant and vacuum-expectation-value at finite volume: in lowest order

The functions in this section return the finite volume corrections to the mass-squared in terms of the quarkmassnf structure for the partially quenched case. It is assumed that all the quarks have the same mass. The label `XXX=SUN,SON,SPN` refers to the three possible patterns of symmetry breaking, $SU(N_F)\times SU(N_F)\to SU(N_F)$, $SU(N_F)\to SO(N_F)$ and $SU(2N_F)\to Sp(2N_F)$ and `nf`$=N_F$ as defined in this way.
The last letter `x` is `b` for the finite volume integrals evaluated using the Bessel function method or `t` when they are evaluated using the theta function method.

The finite volume correction to the mass is defined as in [34] as

$$\Delta^V M^2_{\text{phys}} \equiv M^{2\,V}_{\text{phys}} - M^{2\,V=\infty}_{\text{phys}} = M^{V(4)2} + M^{V(6)2}\,,$$

$$M^{V(6)2} = M_L^{V(6)2} + M_R^{V(6)2}\,, \tag{116}$$

The expressions are defined in terms of lowest order masses and decay constant. `L` is the size of the spatial directions.
`mass=quarkmassnf({Bm1,Bm4},f0,mu,2)`
`double mnfXXXp4PQVx(int nf, quarkmassnf mass, double L)` returns $M^{V(4)2}$
`double mnfXXX6PQVx(int nf, quarkmassnf mass, Liinf Liin, double L)` returns $M^{V(6)2}$
`double mnfXXXp6LPQVx(int nf, quarkmassnf mass, Liinf Liin, double L)` returns $M_L^{V(6)2}$
`double mnfXXXp6RPQVx(int nf, quarkmassnf mass, double L)` returns $M_R^{V(6)2}$

The interface used here is with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L_{11}^r is always included even if it plays no role. In addition the running of the LECs is not correct for `XXX=SON,SPN`.

The finite volume correction to the decay constant is defined as in [34] as

$$\begin{aligned}\Delta^V F_{\text{phys}} \equiv& F_{\text{phys}}^V - F_{\text{phys}}^{V=\infty} = F_0\left(F^{V(4)2} + F^{V(6)2}\right)\,,\\ F^{V(6)} =& F_L^{V(6)} + M_R^{V(6)}\,.\end{aligned} \tag{117}$$

The expressions are defined in terms of lowest order masses and decay constant.
`mass=quarkmassnf({Bm1,Bm4},f0,mu,2)`
`double fnfXXXp4PQVx(int nf, quarkmassnf mass, double L)` returns $F^{V(4)}$
`double fnfXXX6PQVx(int nf, quarkmassnf mass, Liinf Liin, double L)` returns $F^{V(6)}$
`double fnfXXXp6LPQVx(int nf, quarkmassnf mass, Liinf Liin, double L)` returns $F_L^{V(6)}$
`double fnfXXXp6RPQVx(int nf, quarkmassnf mass, double L)` returns $F_R^{V(6)}$

The interface used here is with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L_{11}^r is always included even if it plays no role. In addition the running of the LECs is not correct for `XXX=SON,SPN`.

The vacuum expectation value is defined for a single quark similar to [34] as

$$\begin{aligned}\Delta^V\langle\bar{q}q\rangle_{\text{phys}} \equiv& \langle\bar{q}q\rangle_{\text{phys}}^V - \langle\bar{q}q\rangle_{\text{phys}}^{V=\infty} = -B_0F_0^2\left(\langle\bar{q}q\rangle^{V(4)} + \langle\bar{q}q\rangle^{V(6)}\right)\,,\\ \langle\bar{q}q\rangle^{(6)} =& \langle\bar{q}q\rangle_L^{(6)} + \langle\bar{q}q\rangle_R^{(6)}\,.\end{aligned} \tag{118}$$

The expressions are defined in terms of lowest order masses and decay constant.
`mass=quarkmassnf({Bm1,Bm4},f0,mu,2)`
`double qnfXXXp4PQVx(int nf, quarkmassnf mass, double L)` returns $\langle\bar{q}q\rangle^{V(4)}$
`double qnfXXX6PQVx(int nf, quarkmassnf mass, Liinf Liin, Ki Kiin)` returns $\langle\bar{q}q\rangle^{V(6)}$

`double qnfXXXp6LPQVx(int nf, quarkmassnf mass, Liinf Liin)` returns $\langle\bar{q}q\rangle_L^{V(6)}$
`double qnfXXXp6RPQVx(int nf, quarkmassnf mass)` returns $\langle\bar{q}q\rangle_R^{V(6)}$

The interface used here is with the n_F-flavour ChPT data structures. This is correct for `XXX=SUN` and the results are independent of the subtraction scale as is needed. However, the extra constant L_{11}^r is always included even if it plays no role. In addition the running of the LECs is not correct for `XXX=SON,SPN`.

Defined in `massdecayvevnfPQV.h`, implemented in `massdecayvevnfPQV.cc` and examples of use in `testmassdecayvevnf.cc`.

Acknowledgements

This work is or has been supported, in part, by the Swedish Research Council grants 621-2011-5080, 621-2013-4287 and 2015-04089. I thank all my collaborators in the various applications 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 [17].

A GNU GENERAL PUBLIC LICENSE

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.
For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.
We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.
Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.
Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.
The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

B Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License (”Public License”). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

i **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j Sui Generis Database Rights means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

j You means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

   A. reproduce and Share the Licensed Material, in whole or in part; and

   B. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.
3. Term. The term of this Public License is specified in Section 6(a).
4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.
5. Downstream recipients.

   A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

   B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.
2. Patent and trademark rights are not licensed under this Public License.
3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:

   A. retain the following if it is supplied by the Licensor with the Licensed Material:

   i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

   ii. a copyright notice;

   iii. a notice that refers to this Public License;

   iv. a notice that refers to the disclaimer of warranties;

   v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

   B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

   C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.
3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.
4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

References

[1] J. Bijnens, Eur. Phys. J. C **75** (2015) 1, 27 [arXiv:1412.0887 [hep-ph]].

[2] `http://creativecommons.org/licenses/by/4.0/`.

[3] `http://en.wikipedia.org/wiki/Chiron`

[4] `http://www.gnu.org/licenses/gpl-2.0.htmlhttp://www.gnu.org/licenses/gpl-2.0.html`

[5] `http://www.thep.lu.se/~bijnens/chiron/`

[6] G. 't Hooft, M. J. G. Veltman, Nucl. Phys. **B153** (1979) 365-401.

[7] `http://cernlib.web.cern.ch`

[8] H. Frellesvig, D. Tommasini and C. Wever, JHEP **1603** (2016) 189 doi:10.1007/JHEP03(2016)189 [arXiv:1601.02649 [hep-ph]].

[9] J. Bijnens, E. Boström and T. A. Lähde, JHEP **1401** (2014) 019 [arXiv:1311.3531 [hep-lat]].

[10] J.C.P. Bus and T.J. Dekker, ACM Trans. Math. Software 1 (1975) 330-345.

[11] A. van Doren and L. de Ridder, J. Comput. Appl. Math. 2 (1976) 207-217.

[12] S. Weinberg, Physica A **96** (1979) 327.

[13] J. Gasser and H. Leutwyler, Annals Phys. **158** (1984) 142.

[14] J. Gasser and H. Leutwyler, Nucl. Phys. B **250** (1985) 465.

[15] `http://www.thep.lu.se/∼bijnens/chpt/`

[16] J. Bijnens, Prog. Part. Nucl. Phys. **58** (2007) 521 [hep-ph/0604043].

[17] J. Bijnens and I. Jemos, Nucl. Phys. B **854** (2012) 631 [arXiv:1103.5945 [hep-ph]].

[18] J. Bijnens and G. Ecker, Ann. Rev. Nucl. Part. Sci. **64** (2014) 149 doi:10.1146/annurev-nucl-102313-025528 [arXiv:1405.6488 [hep-ph]].

[19] J. Bijnens, G. Colangelo and G. Ecker, JHEP **9902** (1999) 020 [hep-ph/9902437].

[20] J. Bijnens, G. Colangelo and G. Ecker, Annals Phys. **280** (2000) 100 [hep-ph/9907333].

[21] G. Amorós, J. Bijnens and P. Talavera, Nucl. Phys. B **568** (2000) 319 [hep-ph/9907264].

[22] J. Bijnens and P. Talavera, JHEP **0203** (2002) 046 [hep-ph/0203049].

[23] G. Passarino and M. J. G. Veltman, Nucl. Phys. B **160** (1979) 151.

[24] J. Bijnens, N. Danielsson and T. A. Lähde, Phys. Rev. D **73** (2006) 074509 [hep-lat/0602003].

[25] J. Bijnens and T. A. Lähde, Phys. Rev. D **72** (2005) 074502 [hep-lat/0506004].

[26] J. Bijnens and T. A. Lähde, Phys. Rev. D **71** (2005) 094502 [hep-lat/0501014].

[27] J. Bijnens, N. Danielsson and T. A. Lähde, Phys. Rev. D **70** (2004) 111503 [hep-lat/0406017].

[28] J. Bijnens and T. Rössler, JHEP **1501** (2015) 034 [arXiv:1411.6384 [hep-lat]].

[29] J. Bijnens and K. Ghorbani, Phys. Lett. B **636** (2006) 51 [hep-lat/0602019].

[30] E. Golowich and J. Kambor, Phys. Rev. D **58** (1998) 036004 [hep-ph/9710214].

[31] G. Amorós, J. Bijnens and P. Talavera, Nucl. Phys. B **585** (2000) 293 [Erratum-ibid. B **598** (2001) 665] [hep-ph/0003258].

[32] J. Bijnens and T. Rössler, arXiv:1508.07238 [hep-lat].

[33] J. Bijnens and J. Lu, JHEP **0911** (2009) 116 [arXiv:0910.5424 [hep-ph]].

[34] J. Bijnens and T. Rössler, JHEP **1511** (2015) 017 [arXiv:1509.04082 [hep-lat]].

[35] J. Bijnens and J. Relefors, JHEP **1405** (2014) 015 [arXiv:1402.1385 [hep-lat]].

[36] C. T. Sachrajda and G. Villadoro, Phys. Lett. B **609** (2005) 73 [hep-lat/0411033].

[37] J. Gasser and H. Leutwyler, Nucl. Phys. B **250** (1985) 517.

[38] J. Bijnens and P. Talavera, Nucl. Phys. B **669** (2003) 341 [hep-ph/0303103].

[39] C. Bernard, J. Bijnens and E. Gámiz, Phys. Rev. D **89** (2014) 5, 054510 [arXiv:1311.7511 [hep-lat]].

[40] J. Bijnens, G. Colangelo, G. Ecker, J. Gasser and M. E. Sainio, Phys. Lett. B **374** (1996) 210 doi:10.1016/0370-2693(96)00165-7 [hep-ph/9511397].

[41] J. Bijnens, G. Colangelo, G. Ecker, J. Gasser and M. E. Sainio, Nucl. Phys. B **508** (1997) 263 Erratum: [Nucl. Phys. B **517** (1998) 639] doi:10.1016/S0550-3213(97)80013-2, 10.1016/S0550-3213(97)00621-4, 10.1016/S0550-3213(98)00127-8 [hep-ph/9707291].

[42] U. Burgi, Nucl. Phys. B **479** (1996) 392 doi:10.1016/0550-3213(96)00454-3 [hep-ph/9602429].

[43] U. Burgi, Phys. Lett. B **377** (1996) 147 doi:10.1016/0370-2693(96)00304-8 [hep-ph/9602421].

[44] J. Bijnens, G. Colangelo and P. Talavera, JHEP **9805** (1998) 014 doi:10.1088/1126-6708/1998/05/014 [hep-ph/9805389].

[45] J. Gasser and H. Leutwyler, Phys. Lett. B **184** (1987) 83. doi:10.1016/0370-2693(87)90492-8

[46] G. Colangelo and C. Haefeli, Nucl. Phys. B **744** (2006) 14 doi:10.1016/j.nuclphysb.2006.03.010 [hep-lat/0602017].

Index