BlackHat, a library for one-loop matrix element evaluation

Z. Bern[a], L. Dixon[b], F. Febres Cordero[c], S. Höche[b], H. Ita[d,*], D. Kosower[e], D. Maître[f,g,**], K. Ozeren[a,]

[a] *Department of Physics and Astronomy, UCLA, Los Angeles, CA 90095–1547, USA*
[b] *SLAC National Accelerator Laboratory, Stanford University, Stanford, CA 94309, USA*
[c] *Departamento de Física, Universidad Simón Bolívar, Caracas 1080A, Venezuela*
[d] *Physikalisches Institut, Albert-Ludwigs-Universität Freiburg, D–79104 Freiburg, Germany*
[e] *Institut de Physique Théorique, CEA–Saclay, F–91191 Gif-sur-Yvette cedex, France*
[f] *Theory Division, Physics Department, CERN, CH–1211 Geneva 23, Switzerland*
[g] *Department of Physics, University of Durham, Durham DH1 3LE, UK*

Abstract

In this article we present the BlackHat program library for the evaluation of QCD one-loop matrix elements. The library, written in C++, uses the unitarity method to evaluate the virtual matrix element. This first public release currently supports a variety of processes with up to four final-state QCD partons or electroweak bosons.

Keywords: QCD; vector bosons; jets; LHC; Tevatron.

PROGRAM SUMMARY

Manuscript Title:
Authors:
Program Title: BlackHat
Journal Reference:
Catalogue identifier:
Licensing provisions:
Programming language: C++, *Python interface available*

*Corresponding author.
E-mail address: harald.ita@physik.uni-freiburg.de
**Corresponding author.
E-mail address: daniel.maitre@cern.ch

Computer:
Operating system: Linux
RAM: varying, typically $>$ 1 GB
Keywords: QCD, NLO, vector boson, jets.
Classification:
External routines/libraries: QD
Nature of problem: Calculation of one-loop amplitudes and virtual contributions to squared matrix elements for high-multiplicity final states at colliders

Solution method: C++ implementation of the unitarity method

1. Introduction

This document introduces the BlackHat code library [1, 2, 3], which computes the one-loop amplitudes in QCD required for a variety of next-to-leading order (NLO) predictions of Standard-Model processes. We give a brief overview of its capabilities, explain how to use it, how its external interface is structured, and how to install it. We also give some brief examples of its use in different contexts.

Theoretical predictions of Standard-Model processes for colliders require calculations at NLO in order to achieve a basic quantitative reliability [4]. Such predictions are essential to the physics program at the LHC, and in particular to obtaining a quantitative understanding of backgrounds to potential physics beyond the Standard Model, as well as to a program of precision measurements of Standard-Model resonances such as the top quark and the recently discovered heavy boson [5, 6]. For over two decades, NLO calculations have been carried out at parton-level to fixed order in perturbation theory, both inclusively and exclusively in the number of jets. In the past decade, fixed-order predictions have been matched to parton showers at NLO accuracy, resulting in NLO hadron-level event generators [7, 8]. In the past year, it has become possible to maintain the NLO accuracy for

hadron-level events containing additional jets, within the same merged event sample [9, 10].

NLO calculations require a number of different ingredients, including the same squared tree-level matrix elements required for leading-order (LO) calculations, but to one higher multiplicity, and virtual corrections to the squared matrix elements. The latter require the computation of one-loop amplitudes. The BlackHat code library performs such one-loop computations. It can be applied both to fixed-order NLO calculations and to parton-shower simulations matched to NLO accuracy.

2. Processes

The present release of the BlackHat library can compute the virtual contributions to the squared matrix elements for the following processes [2, 3, 11, 12, 13],

- $jj \to n\,j \qquad n = 2, 3, 4$
- $jj \to W(\to l\nu) + n\,j\,, \qquad n = 0, ..., 3$
- $jj \to \gamma + n\,j\,, \qquad n = 1, ..., 3$
- $jj \to Z/\gamma^*(\to e^+e^-) + n\,j\,, \qquad n = 0, ..., 3$

where j represents either a gluon, a quark, or an antiquark. The vector bosons W and Z decay to a pair of massless leptons, retaining all spin correlations and all effects due to the boson width. The partonic processes above involving the Z boson can be crossed to obtain the virtual contributions to the squared matrix elements needed for e^+e^- collisions. The library is designed to be extended to additional processes, and to higher multiplicities [14, 15].

These partonic subprocesses enable the NLO computation of the following processes at the LHC,

- $pp \to n\,\text{jets}\,, n = 2, 3, 4$
- $pp \to W(\to l\nu) + n\,\text{jets}\,, \qquad n = 0, ..., 3$
- $pp \to \gamma + n\,\text{jets}\,, \qquad n = 1, ..., 3$
- $pp \to Z/\gamma^*(\to e^+e^-) + n\,\text{jets}\,, \qquad n = 0, ..., 3$

3. Method and Implementation

The BlackHat library applies on-shell methods [16, 17, 18, 19, 20, 21, 22, 23, 24, 25], in particular the unitarity method, to the computation of one-loop amplitudes. (For recent reviews see refs. [26, 27, 28].) It first computes *primitive amplitudes* for each non-vanishing helicity configuration of external particles. It assembles these into complete color-ordered amplitudes, and then sums the interference of each one-loop helicity amplitude with the conjugate tree amplitude over colors. The color sum is carried out as a full color sum, or restricted to contributions leading in the number of colors; the choice is specified by the user. The leading-color sum typically provides an excellent approximation to the full result, at much lower cost in computer time. In a fixed-order calculation, the difference (the subleading-color contribution) can be computed to lower statistical accuracy because its overall contribution is much smaller.

Each primitive amplitude is expressed as a sum over Feynman integrals multipled by coefficients, with an added integral-free contribution (the 'rational' contribution). The integrals are given by analytic formulæ. For amplitudes with two incoming partons and up to three final-state objects (partons or electroweak bosons), we have implemented fully analytic formulæ for the integral coefficients and the additional rational terms. For four or more final-state objects, these coefficients are computed numerically from products of tree amplitudes. The required tree amplitudes are computed numerically using on-shell recursion relations [21] or using analytic formulæ derived from $\mathcal{N} = 4$ supersymmetric gauge theory [29].

While performing numerical calculations of coefficients or additional terms, the BlackHat library carries out numerical consistency checks on individual terms [1, 15]. When a term fails such a consistency check, it is recomputed at higher working precision using the `qd` library.

The BlackHat library returns the one-loop virtual contributions to the squared matrix element, divided by the corresponding tree-level squared matrix element and certain prefactors, and also stripped of both strong and electroweak couplings,

$$\widehat{d\sigma}_V^{(1)} \equiv \frac{1}{8\pi\alpha_S\, c_\Gamma(\epsilon)} \frac{d\sigma_V^{(1)}}{d\sigma^{(0)}}\,, \tag{1}$$

where $d\sigma^{(0)}$ is the fully-differential tree-level squared matrix element, $d\sigma_V^{(1)}$ is the real part of the fully differential interference of the one-loop and tree-level

matrix elements, and

$$c_\Gamma(\epsilon) = \frac{1}{(4\pi)^{2-\epsilon}} \frac{\Gamma(1+\epsilon)\Gamma^2(1-\epsilon)}{\Gamma(1-2\epsilon)} \,. \tag{2}$$

The library takes the Cabibbo–Kobayashi–Maskawa (CKM) quark mixing matrix to be diagonal. At present, certain small contributions (top-loop contributions, including the residual top-bottom axial-loop, and with one exception[1], all quark-loop contributions with the Z coupling directly to the quark loop) are omitted (see refs. [20, 11, 15] for more details.)

4. Usage

The BlackHat library will most typically be used inside a larger program. A user seeking to construct such a program will need to understand how to install BlackHat, how to interface to it from within his or her code, and how to link the BlackHat library. The standard interface model uses the Binoth Les Houches Accord (BLHA) [30]. A user seeking to run a program constructed using this interface, or the dedicated SHERPA [31] one, will need to know how to install BlackHat and how to run the code. Public versions of SHERPA starting with version 2.0.β include a built-in interface to BlackHat modeled on BLHA. Interactive calls to the BlackHat library, intended for testing, can be made using the `python` interface.

In this section, we explain how to use the BlackHat interface to SHERPA; we explain usage in a C++ program in sect. 5; that in a FORTRAN program in sect. 6; and that of the `python` interface in sect. 7. We give installation instructions for the BlackHat library in sect. 9. We give an overview of the BLHA interface in sect. 10.

To use BlackHat within SHERPA, the user must install both BlackHat and SHERPA. He or she has two options for how to do this, as described more fully in sect. 9: install BlackHat, and then configure SHERPA to use it; or install SHERPA, and then build BlackHat to be plugged in.

SHERPA will perform an NLO calculation when the following lines are added to the process declaration section in the run-control file `Run.dat`, in between the `Process` and `End process` lines,

[1]The vector-current light-quark loops are included in Z + 2-jet production; they are intrinsically absent at NLO in Z + 1-jet production.

```
Loop_Generator BlackHat;
NLO_QCD_Mode ⟨mode⟩;
NLO_QCD_Part BVIRS;
```

The *mode* variable in the second of these lines should be set to 1 for a fixed-order NLO calculation, and to 3 for an NLO-matched parton-shower calculation. In the former case, SHERPA will switch off the shower by default. In the latter case, SHERPA will enable the shower by default.

The color summation performed for the virtual contribution, discussed in the previous section and explained more fully in sect. 11, can be controlled by a

```
Subdivide_Virtual ⟨color mode⟩;
```

line in the `Process` section. The *color mode* variable can take one of three values: `FullColor` (the default), `LeadingColor`, and `FullMinusLeadingColor`.

5. Example: Linking to a C++ Program

A C++ program calling the BlackHat library will use two interface routines, `OLP_Start` and `OLP_EvalSubProcess`. The program can either include the header file `BH_LH_Interface.h`, or must specify an `extern` declaration for the two interface routines, as shown in the example below. Before evaluating any amplitudes, the interface must be initialized with a specific *contract* using `OLP_Start`. The interface is specified by the BLHA, which we shall explain in greater detail in sect. 10. The contract is embodied in a file which must be supplied to `OLP_Start`. The result for a given subprocess is returned by `OLP_EvalSubProcess`. As explained earlier, this routine computes the interference of the one-loop amplitude with the tree-level amplitude. In principle, the contract file controls whether the quantities computed are to be summed over colors and helicities; in the present release, summation over helicities is the only supported option. A complete sum over colors, a sum restricted to leading-color contributions, or the difference of these two, are the only options currently supported for the color sum. Currently, the only supported return formats are those corresponding to eq. (1), where results are normalized by the squared tree-level amplitude and other prefactors, and stripped of strong and electroweak couplings, which must be multiplied in by the caller. The results are separated into different terms in the Laurent expansion in the dimensional regulator ϵ, where $D = 4-2\epsilon$. The two routines have the following calling conventions,

```
OLP_Start(const char *filename, int *success)
```
`filename` *the name of the contract file*
`success` *will be set to 1 if the contract is valid*

```
OLP_EvalSubprocess(int *process_id, double *momenta,
      double *renormalization_scale,
      double *parameter_list,
      double *result)
```
`process_id` *the id of the desired subprocess, as given in the contract file*
`momenta` *an array of momenta, each containing a four-vector and a mass*
`renormalization_scale` *the renormalization scale in GeV*
`parameter_list` *an array of couplings and other parameters (ignored as* BlackHat *returns normalized values)*
`result` *an array of four* `double`*s, containing the results for the* $1/\epsilon^2$*, the* $1/\epsilon$*, and the* ϵ^0 *terms of the virtual part of the squared matrix element, normalized as in eq. (1); the last element is set to 1*

The following program gives an example of usage of these two routines,

```
#include <iostream>

using namespace std;

extern "C" {
extern void OLP_EvalSubProcess(int* Label,double* Momenta,double *mu,double *param
extern void OLP_Start(const char* filename,int *status);
}

int main(){
        int success;
        OLP_Start("contract_file.lh",&success);
        if (success != 1) { return 1; }
        double result[4];
        double parameters[]={1.0,1.0};
        double momenta[]={
//   E      x      y      z        m
   500.0, 0.0,    0.0,  500.0,    0.0,
   500.0, 0.0,    0.0, -500.0,    0.0,
```

```
   500.0, 0.0, -500.0,    0.0,    0.0,
   500.0, 0.0,  500.0,    0.0,    0.0,
   };

  double mu=500; //
  int label=1;
  OLP_EvalSubProcess(&label,momenta,&mu,parameters,result);

  cout << "1/e^2 : " << result[0] << endl;
  cout << "1/e   : " << result[1] << endl;
  cout << "finite: " << result[2] << endl;

}
```

The program should be compiled and linked with the aid of the `blackhat-config` script,

```
g++ -c -o cpp_program.o cpp_program.cpp
ld cpp_program.o `blackhat-config --libs`
```

An example program is present in the `examples` directory of the BlackHat distribution (which is copied to the `share/blackhat/examples` directory upon installation). We discuss an example of a contract file in sect. 10.

6. Example: Linking to a FORTRAN Program

Calling the BlackHat library from within a FORTRAN program is similar to calling it from within a C++ program; the routine `OLP_Start` must be called to initialize the library, and `OLP_EvalSubProcess` is then called to evaluate virtual contributions. No include file is necessary.

```
      program fortran_example
      implicit none
      integer i,nexternal,status,label
      parameter (nexternal=4)
      double precision pmass(nexternal),p(0:4,nexternal)
      character*16 filename
      double precision couplings(2),mu,virt_wgts(4)

      data (p(i,1),i=0,4)/ 0.5000000E+02 , 0.0000000E+00 ,
```

```
&     0.0000000E+00 , 0.5000000E+02 , 0.0000000E+00 /
 data (p(i,2),i=0,4)/ 0.5000000E+02 , 0.0000000E+00 ,
&     0.0000000E+00 ,-0.5000000E+02 , 0.0000000E+00 /
 data (p(i,3),i=0,4)/ 0.5000000E+02 , 0.1109243E+02 ,
&     0.4448308E+02 ,-0.1995529E+02 , 0.0000000E+00 /
 data (p(i,4),i=0,4)/ 0.5000000E+02 ,-0.1109243E+02 ,
&     -0.4448308E+02 , 0.1995529E+02 , 0.0000000E+00 /

 filename = "contract_file.lh"
 mu =100d0
 label=1

 call OLP_Start(filename//CHAR(0),status)
 call OLP_EvalSubprocess(label,p,mu,couplings,virt_wgts)

 write, virt_wgts

 return
 end
```

A FORTRAN program should be linked to the C libraries in addition to the BLACKHAT library, as follows,

```
gfortran -c -o fortran_example.o fortran_example.f
ld fortran_example.o `blackhat-config --libs` -lc -lstdc++
```

An example program is present in the `examples` directory of the BLACKHAT distribution (which is copied to the `share/blackhat/examples` directory upon installation).

Because FORTRAN is case-insensitive, different compilers have different conventions for naming functions in object files. Some compilers add trailing and/or leading underscores, some capitalize names, whereas other compilers convert names to lower case. We have tried to accomodate all conventions of which we are aware. We encourage readers to inform one of the corresponding authors of any conventions we have missed.

7. Python interface

The BLACKHAT library can also be called from `python`, using an implementation of the BLHA interface described in greater detail in sect. 10. To

enable this interface, the BlackHat `configure` script should be run with the `--enable-LHpythoninterface` option before building the library. This option creates a `python` module called BHLH, which must be loaded. (The BlackHat installation directory must be in the `python` search path, and the BlackHat library must be in the linker's search path.) The module provides the following functions,

- `SignContract(orderFileName,contractFileName)` reads an order file, specified by `orderFileName`, in BLHA format [30] and writes the corresponding contract file to a file specified by `contractFileName`. The filenames may be either absolute or relative to the current working directory.

- `Init(ContractFileName)` initializes the library for use with the contract file named in the argument.

- `BHLH.EvalSubprocess(id,momenta,mu,alphas,alphaEW)` evaluates the virtual correction for the process with the given `id` in the contract file. The momentum arguments are given by `momenta`, and the scale is set to `mu`. The remaining arguments `alphas` and `alphaEW` are present for compatibility, but are ignored by BlackHat, as all results returned are stripped of coupling factors. Each momentum must be a five-tuple, with entries (Energy, x-, y-, z-components of the momentum, mass). The function returns a four-tuple containing as its first three entries the $1/\epsilon^2$, the $1/\epsilon$, and the ϵ^0 terms in the virtual correction to the squared matrix element (the interference of the one-loop amplitude with the tree-level amplitude), divided by the squared tree-level matrix element. The last entry is set to 1.

The following code fragment gives an example of BlackHat use from within python,

```
import BHLH

BHLH.SignContract('order_file.lh','contract_file.lh')
BHLH.Init('contract_file.lh')

momenta=[
# E  x  y  z    m
  1, 0, 0, 1,   0,
```

```
  1, 0, 0,-1,   0,
  1, 0, 1, 0,   0,
  1, 0,-1, 0,   0
]

double,single,finite,tree=BHLH.EvalSubprocess(1,momenta,10,0,0)

print 'A=%s/eps^2 + %s/eps + %s' % (double,single,finite)
```

An example program is present in the `examples` directory of the BlackHat distribution (which is copied to the `share/blackhat/examples` directory upon installation).

8. Detailed SHERPA Examples

The following run-control file directs SHERPA to perform a parton-level NLO calculation of inclusive W production at 8 TeV in pp collisions using the BlackHat library:

```
(run){
   # technical parameters
   ME_SIGNAL_GENERATOR Amegic BlackHat;
  EVENT_GENERATION_MODE W;
   # physics parameters
   BEAM_1 2212; BEAM_ENERGY_1 4000;
   BEAM_2 2212; BEAM_ENERGY_2 4000;
}(run);
(processes){
   Process 93 93 -> 11 -12;
   NLO_QCD_Mode 1; NLO_QCD_Part BVIRS;
   Loop_Generator BlackHat;
   Scales VAR{sqr(80.419)};
   Order_EW 2;
   End process;
}(processes);
```

The renormalization and factorization scales are set to the W mass, taken here to be 80.419 GeV. The lines containing the `Loop_Generator`, `NLO_QCD_Mode`, and `NLO_QCD_Part` commands switch on the NLO aspects of the calculation, as discussed in sect. 4. We refer the user to the SHERPA manual located at

`http://sherpa.hepforge.org/doc/SHERPA-MC-1.4.2.html` for a description of the control lines not discussed in sect. 4. SHERPA should be run with this control file in a standard fashion. This example file is present in the `examples` directory of the BlackHat distribution (which is copied to the `share/blackhat/examples` directory upon installation).

A similar but more complicated example is offered by the following run-control file, which directs SHERPA to perform a parton-level NLO calculation of inclusive $W^+ + 2$-jet production at 8 TeV in pp collisions using the BlackHat library:

```
(run){
  # technical parameters
  ME_SIGNAL_GENERATOR Amegic Comix BlackHat;
  EVENT_GENERATION_MODE W;
  DIPOLE_ALPHA 0.03;
  # physics parameters
  BEAM_1 2212; BEAM_ENERGY_1 4000;
  BEAM_2 2212; BEAM_ENERGY_2 4000;
  SCALES VAR{sqr(sqrt(H_T2)-PPerp(p[2])-PPerp(p[3])+MPerp(p[2]+p[3]))};
}(run);
(processes){
  ### The Born, virtual and integrated subtraction piece
  Process 93 93 -> 11 -12 93 93;
  NLO_QCD_Mode 1; NLO_QCD_Part BVI;
  Loop_Generator BlackHat;
  Order_EW 2;
  End process;
  ### The real emission piece and subtractions
  Process 93 93 -> 11 -12 93 93;
  NLO_QCD_Mode 1; NLO_QCD_Part RS;
  ME_Generator Comix;
  Order_EW 2;
  End process;
}(processes);
(selector){
  FastjetFinder antikt 2 20 0 0.4;
}(selector);
```

This example uses $\hat{H}'_T$, defined in ref. [14] for both the renormalization and factorization scales. It makes use of the anti-k_T algorithm [32] to define jets, using the FastJet library [33, 34]. In order to use this library, the `--enable-fastjet=`⟨path to FastJet⟩ option must be give to the `configure` script when installing SHERPA.

9. Installation

Installation of the BlackHat code library proceeds as follows:

1. Install the `qd` library from its distribution page,

   ```
   wget http://crd.lbl.gov/~dhbailey/mpdist/qd-2.3.13.tar.gz
   ```

   unzip, untar, and follow the package's installation instructions. Use the configure option:

   ```
   --enable-shared
   ```

   Make sure that the shared libraries have been built before attempting to install `qd`. In this regard, note that the `qd` build scripts may look for a C++ compiler other than `g++`, and depending on which one is chosen, the `qd` build may fail to build the shared libraries. If the scripts select a compiler other than `g++`, one should specify the use of `g++` (or another compiler which supports shared libraries) explicitly using the additional argument `CXX=g++` on the `configure` command line.
2. Install the BlackHat library from its distribution page,

   ```
   wget http://www.hepforge.org/downloads/blackhat/blackhat-1.0.0.tgz
   ```

   Extract the sources,

   ```
   tar -xzf blackhat-1.0.0.tar.gz
   cd blackhat-1.0.0
   ```

   Configure using the following options (in addition to standard options such as `--prefix` if required):

`--enable-sherpaplugin=`⟨path to SHERPA⟩	*compiles the* BlackHat SHERPA *plugin*
`--with-QDpath=prefix`	*if the* `qd` *library is installed in a non-standard place*
`--enable-LHpythoninterface`	*compiles the python interface*

The installation path can be chosen using the standard `configure` option `--prefix=`⟨installation path⟩.

3. Build the BlackHat library by executing `make` in the working directory (from which the `configure` script was run).
4. Install the BlackHat library by executing `make install`.

As mentioned earlier in sect. 4, in order to use BlackHat with SHERPA, both must be installed. There are two ways of doing this[2] ,

1. Install the BlackHat library, and then build a post-2.0.β version of SHERPA, using the configure option `--enable-blackhat=`⟨top-level BlackHat installation path⟩ in building the latter. The user then runs SHERPA without any special command-line arguments, but with additional lines in SHERPA run-control file `Run.dat` as described in sect. 4;
2. Install a post-2.0.β version of SHERPA, and then install the BlackHat library, using the configure option `--enable-sherpaplugin`, as explained above. When running SHERPA, the user must then either supply an additional command-line argument,

   ```
   SHERPA_LDADD=BH_Sherpa_Interface
   ```

   or add an additional line of the same form to the SHERPA run-control file `Run.dat`, along with additional lines in the run-control file as described earlier in sect. 4. The BlackHat library name as given should not include the leading `lib` or trailing `.so`; the library must be in the user's library search path (usually `$LD_LIBRARY_PATH`; the required directory is in the line returned by the `blackhat-config` script described below)

Once the installation is complete, there will be a `bin` directory in the BlackHat installation directory. It contains three programs,

1. `blackhat-config` is a tool used in compiling and linking programs using the BlackHat library,

   - `blackhat-config --libs` returns a set of compiler or linker flags adding the path where the libraries are installed,

[2] In principle, there is a third option, using external BLHA order and contract files, but there is no real advantage over the native implementations described above, and a description of this option is beyond the scope of the present document.

- `blackhat-config --include` returns a set of compiler flags adding the path where the header files are installed.

2. `LH_reader` reads BLHA order files and then creates contract files, as described below in sect. 10,

   ```
   LH_reader order_file_name contract_file_name
   ```

3. `dataInstall` is a simple script that installs data files for the evaluation of processes. When a new process within the classes listed in sect. 2 becomes available, the corresponding data files should be installed to activate them.

10. Interface

BlackHat supports the BLHA interface mechanism described in ref. [30]. We review the essential elements here; you are encouraged to read the just-cited reference for further details.

The mechanism envisages two codes collaborating to calculate NLO corrections for a given process or set of processes. The first code — here called the One-Loop Provider (OLP) — computes the virtual corrections to squared matrix elements, and supplies them to another code, here called the User (U) program, which supplies the phase-space points at which evaluations of the virtual corrections are required. The U program is typically a parton-level integrator or a parton-shower Monte Carlo, but the code model is more general.

The mechanism has two stages, in order to be compatible with code-generating programs. In the first phase, the two codes exchange information about parameters and capabilities. Only in the second stage do the codes perform actual calculations. In more detail, in the first stage the U program puts a list of all subprocesses for which virtual corrections are required into an *order file*, along with various options for what precise quantities are to be calculated, and how the calculations are to be performed. Some options are present in the BLHA standard, while others are BlackHat-specific. The OLP reads the order file, and if it can provide all requested calculations in the requested manner, writes out information that the U program will need in order to perform the calculation for each and every subprocess requested. The OLP writes this information in a *contract file*. This file is then used to initialize the OLP at the beginning of the second phase.

The following excerpt is an example of an order file,

```
# BLHA options
MatrixElementSquareType CHsummed
CorrectionType QCD
MassiveParticleScheme OnShell
IRsubtractionMethod None
OperationMode NormalizedByBorn
SubdivideProcess No
IRregularisation CDR

# BH options
Z_mass 91.1876
W_mass 80.385
Z_width 2.4952
W_width 2.085
COLOR_MODE full_color

# Processes required
11 -11 -> 1 -1
11 -11  -> 1 -1
```

At the end of the first stage, BlackHat then returns the following contract file,

```
# BLHA options
MatrixElementSquareType CHsummed                  | OK
CorrectionType QCD               | OK
MassiveParticleScheme OnShell            | OK
IRsubtractionMethod None                 | OK
OperationMode NormalizedByBorn           | OK
SubdivideProcess No              | OK
IRregularisation CDR             | OK

# BH options
Z_mass 91.1876           | OK
W_mass 80.385            | OK
Z_width 2.4952           | OK
W_width 2.085            | OK
COLOR_MODE full_color            | OK
```

```
# Processes required
11 -11 -> 1 -1  | 1 1
11 -11  -> 1 -1         | 1 2

#
# options
#
# MatrixElementSquareType: CHsummed
# CorrectionType: QCD
# IRregularisation: CDR
# IRsubtractionMethod: None
# MassiveParticleScheme: OnShell
# OperationMode: NormalizedByBorn
# SubdivideProcess: No
#
```

The `BlackHat` library currently supports the following BLHA options and values

`MatrixElementSquareType` currently, only `CHsummed` (color- and helicity-summed) is supported

`CorrectionType` currently, only `QCD` is supported

`MassiveParticleScheme` currently, only `OnShell` is supported

`IRsubtractionMethod` currently, only `None` is supported

`SubdivideProcess` currently, only `No` is supported

`IRregularisation` `CDR` (conventional dimensional regularization) and `tHV` ('t Hooft–Veltman) are supported

`OperationMode` currently, only `NormalizedByBorn`, specifying that results are returned according to eq. 1. Modifying the color sum should be done via the BlackHat option `Color_Mode`, described below.

The following options are not currently supported or are ignored

`ModelFile`

`SubdivideSubprocess`

In addition, the BlackHat library also supports the additional options,

`Z_mass` Mass of the Z boson in GeV

`Z_width` Width of the Z boson in GeV

`W_mass` Mass of the W boson in GeV

`W_width` Width of the W boson in GeV

`sin_th_2` The value of $\sin^2\theta_W$

`Color_Mode` Currently, three options are supported for computations with with four or more final-state objects: `FullColor`, which specifies that the return values comes from a complete color sum of the virtual inteference; `LeadingColor`, which restricts the color sum to terms leading in the number of colors; and `FullMinusLeadingColor`, which specifies that the return value includes only subleading-color terms, precisely those not included in the `LeadingColor` return value

11. Efficiency Considerations

The computation of one-loop corrections is considerably more demanding of computer resources than that of the basic squared matrix elements needed for an LO estimate. (The computation of real-emission corrections is likewise much more computationally demanding.) This makes it worthwhile to consider approximations to, or decompositions of, the virtual corrections that retain all essential physics to high accuracy but reduce the computational load signficantly.

At present, the BlackHat library supports one decomposition and associated approximation that can reduce the computational effort by an order of magnitude in processes with four or more final-state objects. (The decomposition is not currently supported for processes with fewer final-state objects, where it isn't needed for efficiency purposes.) This is the decomposition of the virtual corrections into terms that are leading in the number of colors, and terms that are subleading. (The decomposition refers to the scaling in a theoretical limit where the number of colors, $N_c = 3$ in QCD, is taken to infinity.) Such a decomposition is not unique; the form currently adopted in BlackHat is described in refs. [35, 13].

The leading-color approximation for the virtual contributions can be used in both fixed-order and NLO-matched parton-shower calculations. One expects the remaining subleading-color contributions to be of $\mathcal{O}(10\%)$ in QCD. For $W, Z + n$-jet production with up to three jets, this approximation is considerably better: it is known to be valid to 3% once all contributions are combined into a total cross section [35, 3].

In fixed-order calculations, one can use the decomposition to greatly reduced the required computer time for a calculation even without making any approximation. One calculates the leading-color and subleading-color virtual contributions separately; because the subleading-color contributions give a small contribution, they can be computed with much lower statistics. Typically, their computation for a given phase-space point is much lengthier than that of the leading-color ones, so this reduction greatly reduces the overall computer time.

In a similar way, splitting up the computation of the different contributions (Born, virtual, integrated subtraction terms, and subtracted real-emssion) within a fixed-order computation using SHERPA also offers the user the possibility of varying the number of phase-space points used according to both the computer time and the estimated statistical error. (The quantity to minimize is $\sigma\sqrt{\tau}$, where τ is the computer time, and σ the statistical error.) As an example, in the study of W + 3-jet production [2, 3], we used $2 \cdot 10^6$ phase-space points for the leading-color virtual contributions, 10^5 for the subleading ones, and 10^7 for both the born and subtracted real-emission contributions.

12. LHreader

The `LHreader` program is part of the BlackHat package. It can be used to generate BLHA contract files. It takes two command-line arguments, the names of the input order file and the output contract file. If the contract file exists, it will be overwritten,

```
LHreader order_file.lh contract_file.lh
```

13. Validation

We have applied a variety of cross checks to ensure the correctness of the amplitude results produced by the BlackHat library. A subset of the

tests are included in the release, and may be used to validate the installation of the library. The tests include comparisons against published analytic results for lower-point amplitudes; comparisons against other one-loop codes; collinear-factorization tests; comparisons of results using different algorithms; and comparisons against higher-precision results. In addition to cross checks of the matrix elements, we compared cross sections after high-statistics integrations over phase space.

We have compared numerical results for primitive amplitude from BlackHat against available analytic expressions from published results. We have compared most amplitudes with up to three final-state objects, and a selection of amplitudes with four final-state objects. All these comparisons show agreement. We have also made sure that all infrared singularities give the expected pole structure [36] for the double and single poles in the dimensional regulator ϵ. We used Catani–Seymour dipoles [37] implemented as in SHERPA [31, 38, 39] for this numerical check.

We have also compared the virtual corrections to results from other one-loop codes (in particular: HELAC-1Loop [40, 41], and Rocket [42] with S. Frixione's help) at selected phase-space points. This includes all subprocesses for W, Z+3-jet and four-jet production. Badger et al. have also compared four-jet matrix elements at selected phase-space points. The results agree, which serves to increase confidence in all codes in any given comparison, as it is unlikely that errors would give identical discrepancies on both sides of a comparison.

We have checked the collinear limits of BlackHat amplitudes extensively. In collinear limits, amplitudes are related to lower-point amplitudes multiplied by universal splitting amplitudes. In each case, we compute the original n-point amplitude and the collinear approximation using the splitting and $(n-1)$-point amplitudes at a sequence of points approaching the collinear limit. We compute both with a very large number of digits (roughly one thousand decimal digits, using a private version), in order to avoid effects of round-off error and potential numerical instabilities. The numerical convergence of the result to the approximation in the limit (checked to 35 decimal digits for the integral-containing terms, and 15 digits for the amplitude as a whole) is a very strong consistency check on the original n-point amplitude, as many different terms in the numerical computation have to combine in the right way to obtain the correct collinear behavior. We then repeat this check for the $(n-1)$-point amplitude, looking at its collinear limits. We performed these tests for randomly-chosen collinear-approaching

sets of momenta. We do not restrict them to kinematic sectors corresponding to $2 \to n$ scattering, but include other regions such as $3 \to n-1$ scattering, and so forth.

For example, we have compared the collinear limits of six-gluon amplitudes to five-gluon amplitudes, and collinear limits of five-gluon amplitudes to four-gluon amplitudes. This chain,

$$6\,g \to 5\,g \to 4\,g, \tag{3}$$

checks the consistency of an entire family of scattering amplitudes.

Certain contributions (the so-called 'rational terms') can be computed using two entirely different approaches within the framework of on-shell methods: on-shell recursion [21, 22] and D-dimensional unitarity [18, 19, 24, 25]. Independent terms in the two approaches are entirely unrelated. We have compared results obtained using the two different approaches for a variety of amplitudes. The agreement between the results validates the code for both approaches, as again it would be essentially impossible for both codes to produce the same erroneous results.

We have also confirmed the numerical stability of results produced by the library. To do so, we compared values of the virtual corrections to reference evaluations using a much larger number of digits (roughly 60 decimal digits). In 99% of randomly-chosen phase-space points, we find agreement to better than five digits. These checks also validate the instability-detection and -repair system discussed in sect. 3.

We have published total and differential cross sections for all process families provided [2, 3, 14, 11, 15, 12, 43, 13]. Several results have since been validated by other collaborations [42, 44]. In addition, we have compared the W, Z+2-jet cross sections to results from MCFM [45].

The BlackHat tarball includes a basic set of tests that allow the user to validate an installation. These tests may be found in `tests`. To run them, issue the command

```
make check
```

Some of the tests are quite lengthy.

Acknowledgments

The BlackHat project has been supported by the US Department of Energy under contracts DEFG0391ER40662 and DEAC0276SF00515. DAKs

research is supported by the European Research Council under Advanced Investigator Grant ERCAdG228301. DMs work was supported by the Research Executive Agency (REA) of the European Union under the Grant Agreement number PITN–GA–2010–264564 (LHCPhenoNet). The work of KO and SH was partly supported by a grant from the US LHC Theory Initiative through NSF contract PHY–0705682. This research also used resources of Academic Technology Services at UCLA.

References

[1] C. F. Berger, Z. Bern, L. J. Dixon, F. Febres Cordero, D. Forde, H. Ita, D. A. Kosower, D. Maître, *An Automated Implementation of On-Shell Methods for One-Loop Amplitudes*, Phys. Rev. D78 (2008) 036003. arXiv:0803.4180, doi:10.1103/PhysRevD.78.036003.

[2] C. F. Berger, Z. Bern, L. J. Dixon, F. Febres Cordero, D. Forde, T. Gleisberg, H. Ita, D. A. Kosower, D. Maître, *Precise Predictions for W + 3 Jet Production at Hadron Colliders*, Phys. Rev. Lett. 102 (2009) 222001. arXiv:0902.2760, doi:10.1103/PhysRevLett.102.222001.

[3] C. F. Berger, Z. Bern, L. J. Dixon, F. Febres Cordero, D. Forde, T. Gleisberg, H. Ita, D. A. Kosower, D. Maître, *Next-to-Leading Order QCD Predictions for W + 3-Jet Distributions at Hadron Colliders*, Phys. Rev. D80 (2009) 074036. arXiv:0907.1984, doi:10.1103/PhysRevD.80.074036.

[4] Z. Bern, et al., *The NLO Multileg Working Group: Summary Report*, Proceedings of the Fifth Les Houches Workshop on Physics at TeV Colliders (2008) 1–120. arXiv:0803.0494.

[5] G. Aad, et al., *Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC*, Phys. Lett. B716 (2012) 1–29. arXiv:1207.7214, doi:10.1016/j.physletb.2012.08.020.

[6] S. Chatrchyan, et al., *Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC*, Phys. Lett. B716 (2012) 30–61. arXiv:1207.7235, doi:10.1016/j.physletb.2012.08.021.

[7] S. Frixione, B. R. Webber, *Matching NLO QCD Computations and Parton Shower Simulations*, JHEP 0206 (2002) 029, arXiv:hep-ph/0204244.

[8] S. Frixione, P. Nason, C. Oleari, *Matching NLO QCD computations with Parton Shower simulations: the POWHEG method*, JHEP 0711 (2007) 070. arXiv:0709.2092, doi:10.1088/1126-6708/2007/11/070.

[9] Siegert, Frank and Höche, Stefan and Krauss, Frank and Schönherr, Marek, *W + n-Jet Predictions at NLO Matched with a Parton Shower*. arXiv:1206.4873.

[10] R. Frederix, S. Frixione, *Merging meets matching in MC@NLO*. arXiv:1209.6215.

[11] C. F. Berger, Z. Bern, L. J. Dixon, F. Febres Cordero, D. Forde, T. Gleisberg, H. Ita, D. A. Kosower, D. Maître, *Next-to-Leading Order QCD Predictions for Z, γ^* + 3-Jet Distributions at the Tevatron*, Phys. Rev. D82 (2010) 074002. arXiv:1004.1659.

[12] Bern, Z. and Diana, G. and Dixon, L. J. and Febres Cordero, F. and Höche, S. and Ita, H. and Kosower, D. A. and Maître, D. and Ozeren, K. J., *Driving Missing Data at Next-to-Leading Order*, Phys. Rev. D84 (2011) 114002. arXiv:1106.1423, doi:10.1103/PhysRevD.84.114002.

[13] Bern, Z. and Diana, G. and Dixon, L. J. and Febres Cordero, F. and Höche, S. and Ita, H. and Kosower, D. A. and Maître, D. and Ozeren, K., *Missing Energy and Jets for Supersymmetry Searches*. arXiv:1206.6064.

[14] C. F. Berger, Z. Bern, L. J. Dixon, F. Febres Cordero, D. Forde, T. Gleisberg, H. Ita, D. A. Kosower, D. Maître, *Precise Predictions for W + 4 Jet Production at the Large Hadron Collider*, Phys. Rev. Lett. 106 (2011) 092001. arXiv:1009.2338, doi:10.1103/PhysRevLett.106.092001.

[15] H. Ita, Z. Bern, L. Dixon, F. Febres Cordero, D. Kosower, D. Maître, *Precise Predictions for $Z + 4$ Jets at Hadron Colliders*, Phys. Rev. D85 (2012) 031501. arXiv:1108.2229.

[16] Z. Bern, L. J. Dixon, D. C. Dunbar, D. A. Kosower, *One-Loop n-Point Gauge-Theory Amplitudes, Unitarity and Collinear Limits*, Nucl.

Phys. B425 (1994) 217–260, arXiv:hep-ph/9403226. doi:10.1016/0550-3213(94)90179-1.

[17] Z. Bern, L. J. Dixon, D. C. Dunbar, D. A. Kosower, *Fusing Gauge-Theory Tree Amplitudes into Loop Amplitudes*, Nucl. Phys. B435 (1995) 59–101, arXiv:hep-ph/9409265. doi:10.1016/0550-3213(94)00488-Z.

[18] Z. Bern, A. G. Morgan, *Massive Loop Amplitudes from Unitarity*, Nucl. Phys. B467 (1996) 479–509, arXiv:hep-ph/9511336. doi:10.1016/0550-3213(96)00078-8.

[19] Z. Bern, L. J. Dixon, D. C. Dunbar, D. A. Kosower, *One-Loop Self-Dual and $\mathcal{N} = 4$ Super-Yang–Mills*, Phys. Lett. B394 (1997) 105–115, arXiv:hep-th/9611127. doi:10.1016/S0370-2693(96)01676-0.

[20] Z. Bern, L. Dixon, D. Kosower, *One Loop Amplitudes for e^+e^- to Four Partons*, Nucl. Phys. B513 (1998) 3, arXiv:hep-ph/9708239.

[21] R. Britto, F. Cachazo, B. Feng, E. Witten, *Direct Proof of Tree-Level Recursion Relation in Yang-Mills Theory*, Phys. Rev. Lett. 94 (2005) 181602, arXiv:hep-th/0501052. doi:10.1103/PhysRevLett.94.181602.

[22] C. F. Berger, Z. Bern, L. J. Dixon, D. Forde, D. A. Kosower, *Bootstrapping One-Loop QCD Amplitudes with General Helicities*, Phys. Rev. D74 (2006) 036009, arXiv:hep-ph/0604195. doi:10.1103/PhysRevD.74.036009.

[23] G. Ossola, C. G. Papadopoulos, R. Pittau, *Reducing Full One-Loop Amplitudes to Scalar Integrals at the Integrand Level*, Nucl. Phys. B763 (2007) 147–169, arXiv:hep-ph/0609007. doi:10.1016/j.nuclphysb.2006.11.012.

[24] C. Anastasiou, R. Britto, B. Feng, Z. Kunszt, P. Mastrolia, *D-Dimensional Unitarity Cut Method*, Phys. Lett. B645 (2007) 213–216, arXiv:hep-ph/0609191. doi:10.1016/j.physletb.2006.12.022.

[25] R. K. Ellis, W. T. Giele, Z. Kunszt, K. Melnikov, *Masses, Fermions and Generalized D-Dimensional Unitarity*, Nucl. Phys. B822 (2009) 270–282. arXiv:0806.3467, doi:10.1016/j.nuclphysb.2009.07.023.

[26] R. K. Ellis, Z. Kunszt, K. Melnikov, G. Zanderighi, *One-loop calculations in quantum field theory: from Feynman diagrams to unitarity cuts*, Phys. Rept. 518 (2012) 141–250. arXiv:1105.4319, doi:10.1016/j.physrep.2012.01.008.

[27] H. Ita, *Susy Theories and QCD: Numerical Approaches*, J. Phys. A44 (2011) 454005. arXiv:1109.6527, doi:10.1088/1751-8113/44/45/454005.

[28] R. Britto, *Loop Amplitudes in Gauge Theories: Modern Analytic Approaches*, J. Phys. A44 (2011) 454006. arXiv:1012.4493, doi:10.1088/1751-8113/44/45/454006.

[29] L. J. Dixon, J. M. Henn, J. Plefka, T. Schuster, *All Tree-Level Amplitudes in Massless QCD*, JHEP 1101 (2011) 035. arXiv:1010.3991, doi:10.1007/JHEP01(2011)035.

[30] T. Binoth, et al., *A Proposal for a Standard Interface Between Monte Carlo Tools and One-Loop Programs*, Comput. Phys. Commun. 181 (2010) 1612–1622. arXiv:1001.1307, doi:10.1016/j.cpc.2010.05.016.

[31] Gleisberg, T. and Höche, Stefan. and Krauss, F. and Schönherr, M. and Schumann, S. and Siegert, F. and Winter, J., *Event generation with SHERPA 1.1*, JHEP 0902 (2009) 007. arXiv:0811.4622, doi:10.1088/1126-6708/2009/02/007.

[32] M. Cacciari, G. P. Salam, G. Soyez, *The Anti-k(t) jet clustering algorithm*, JHEP 0804 (2008) 063. arXiv:0802.1189, doi:10.1088/1126-6708/2008/04/063.

[33] M. Cacciari, G. P. Salam, *Dispelling the N^3 myth for the k_t jet-finder*, Phys. Lett. B641 (2006) 57–61, arXiv:hep-ph/0512210. doi:10.1016/j.physletb.2006.08.037.

[34] M. Cacciari, G. P. Salam, G. Soyez, *FastJet User Manual*, Eur. Phys. J. C72 (2012) 1896. arXiv:1111.6097, doi:10.1140/epjc/s10052-012-1896-2.

[35] H. Ita, K. Ozeren, *Colour Decompositions of Multi-Quark One-Loop QCD Amplitudes*, JHEP 1202 (2012) 118. arXiv:1111.4193, doi:10.1007/JHEP02(2012)118.

[36] S. Catani, *The Singular Behavior of QCD Amplitudes at Two Loop Order*, Phys. Lett. B427 (1998) 161–171, arXiv:hep-ph/9802439. doi:10.1016/S0370-2693(98)00332-3.

[37] S. Catani, M. Seymour, *A General Algorithm for Calculating Jet Cross-Sections in NLO QCD*, Nucl. Phys. B485 (1997) 291–419, arXiv:hep-ph/9605323. doi:10.1016/S0550-3213(96)00589-5.

[38] F. Krauss, R. Kuhn, G. Soff, *AMEGIC++ 1.0: A Matrix element generator in C++*, JHEP 0202 (2002) 044, arXiv:hep-ph/0109036.

[39] T. Gleisberg, F. Krauss, *Automating dipole subtraction for QCD NLO calculations*, Eur. Phys. J. C53 (2008) 501–523. arXiv:0709.2881, doi:10.1140/epjc/s10052-007-0495-0.

[40] A. van Hameren, C. Papadopoulos, R. Pittau, *Automated one-loop calculations: A Proof of concept*, JHEP 0909 (2009) 106. arXiv:0903.4665, doi:10.1088/1126-6708/2009/09/106.

[41] G. Bevilacqua, M. Czakon, M. Garzelli, A. van Hameren, A. Kardos, C. G. Papadopoulos, R. Pittau, M. Worek, *HELAC-NLO*. arXiv:1110.1499.

[42] R. K. Ellis, K. Melnikov, G. Zanderighi, $W + 3$ *Jet Production at the Tevatron*, Phys. Rev. D80 (2009) 094002. arXiv:0906.1445, doi:10.1103/PhysRevD.80.094002.

[43] Bern, Z. and Diana, G. and Dixon, L. J. and Febres Cordero, F. and Höche, S. and Kosower, D. A. and Ita, H. and Maître, D. and Ozeren, K., *Four-Jet Production at the Large Hadron Collider at Next-to-Leading Order in QCD*, Phys. Rev. Lett. 109 (2011) 042001. arXiv:1112.3940, doi:10.1103/PhysRevLett.109.042001.

[44] S. Badger, B. Biedermann, P. Uwer, V. Yundin, *Numerical Evaluation of Virtual Corrections to Multi-Jet Production in Massless QCD*. arXiv:1209.0100.

[45] J. M. Campbell, R. K. Ellis, *Next-to-Leading Order Corrections to $W + 2$ jet and $Z + 2$ Jet Production at Hadron Colliders*, Phys. Rev. D65 (2002) 113007, arXiv:hep-ph/0202176. doi:10.1103/PhysRevD.65.113007.