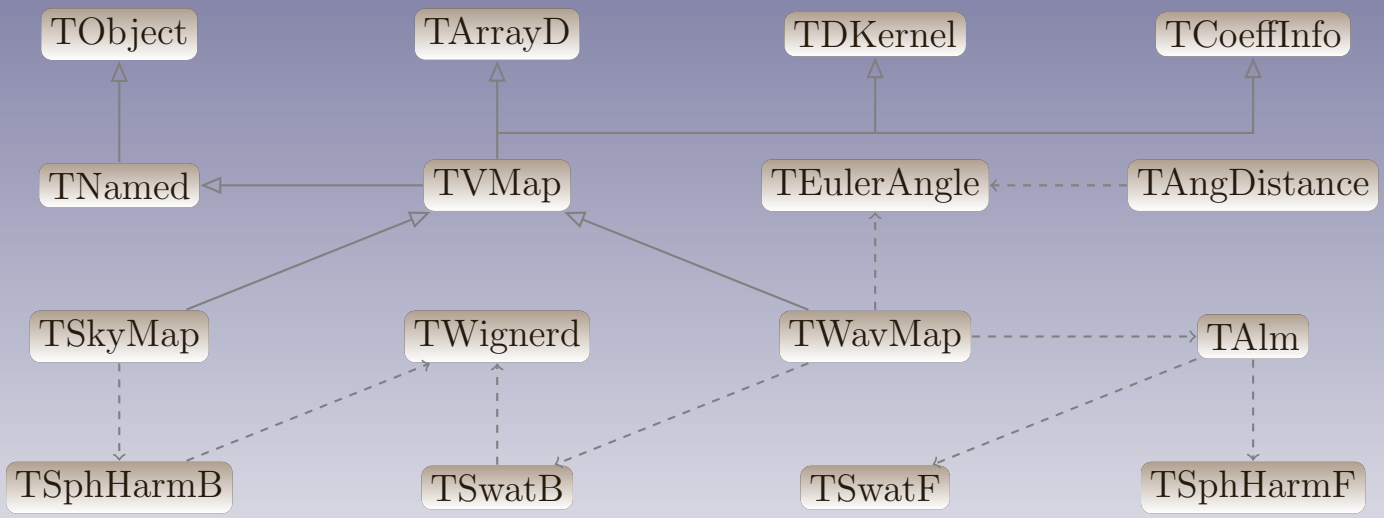

SWAT

The Spherical Wavelet Analysis Tool

Manual for version 2.31

```
Double_t TWignerd::Delta(Int_t m,Int_t n) const
{
  if (m < 0) {
    if (n < 0) {
      if (m >= n)
        return fMatrix[fIndex(-n,-m)];
      return Even(n-m) ? fMatrix[fIndex(-m,-n)]: -fMatrix[fIndex(-m,-n)];
     } else {
      if (n <= Abs(m))
        return Even(fL-n) ? fMatrix[fIndex(-m,n)]: -fMatrix[fIndex(-m,n)];
      return Even(fL-m) ? fMatrix[fIndex(n,-m)]: -fMatrix[fIndex(n,-m)];
     }
  }
    if (n < 0) {
      if (m >= Abs(n))
        return Even(fL-m) ? fMatrix[fIndex(m,-n)]: -fMatrix[fIndex(m,-n)];
      return Even(fL-n) ? fMatrix[fIndex(-n,m)]: -fMatrix[fIndex(-n,m)];
    } else {
      if (m >= n)
        return fMatrix[fIndex(m,n)];
      return Even(n-m) ? fMatrix[fIndex(n,m)]: -fMatrix[fIndex(n,m)];
   }
}
```

Contents

1 Introduction

The software SWAT is a package I have written from scratch for my PhD. project. It evolved from a simple ROOT macro and got bigger and bigger, since the code may be useful for other researchers I decided to document it and make it public. The code is an C++ implementation of the spherical wavelet transform as presented in [**?**]. Some attractive features of SWAT are:

- ROOT is the only prerequisite (with module FFTW enabled).
- You can load all the code in a ROOT session and call SWAT code in your ROOT macro.
- You can save SWAT objects to .root files, which make it very convenient way of exchanging,
- Interface to CRPropa and Pierre Auger Herald data. skymaps, graphs, spherical harmonics etc.

The SWAT package includes some parts of the Healpix code. There are two reasons why I decided to include it here, instead of just link against Healpix libraries. The first is that I (maybe still) do not need all Healpix routines and support to the fits format, and the second is that I usually need shared libraries the call Healpix code in a ROOT session, which is not built by Healpix build system since most of its code is C++ templates.
SWAT is being developed with financial support from CAPES. I also would like to acknowledge UNICAMP, the brasilian Pierre Auger Group and the Pierre Auger Group at university of Wuppertal, with special thanks to Nils Niertenhöfer.

2 Installation

As a prerequisite to install SWAT you need ROOT installed with FFTW module enabled, the configure script will complain if they are not installed. SWAT uses

autotools to generate the configure and Makefile files, so you can expect all the standard configure options and makefile targets.
To use some of the features of the package, I am assuming you are able to build shared libraries on your platform. To easy the task of loading swat libraries the macro load.C, macro will be installed on prefix/share/swat. To load SWAT libraries in ROOT's C++ interpreter you just have to copy this macro to your working directory, or just add it to your code .rootlogon.C macro.
You should also be able to generate documentation in html format with the macro code prefix/share/swat/makehtml.C.

3 What can I do with SWAT?

Swat can be used as a general purpose program for analysis of functions that live on the Sphere. In this section though, we explain only how to find filamentary structures on the spherical data. At the moment it is able to automatically recognize Herald and CRPropa 3D simulations.

4 How to use SWAT

5 High quality graphs using PGFPlots

You may benefit from using SWAT if you want to do some of these things.

```
// Author: Marcelo Zimbres Silva <mailto:mzimbres@gmail.com>

/* Copyright (C) 2009, 2010, 2011 Marcelo Zimbres Silva
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
 */

#include <iostream>
#include <complex>

// ROOT

#include "TMath.h"
#include "TMatrixD.h"
```

```
// SWAT

#include "TWignerd.h"

ClassImp(TWignerd);

using namespace TMath;
using namespace std;

 ////////////////////////////////////////////////////////////////////////
 //
//
 // Calculation of the fourier coefficients of wigner d functions
//
 // using recurrence.
//
 //
//
 // Calculation of spherical harmonics and Wigner d functions by FFT. //
 // Article: Acta Crystallographica Section A, Foundations of
//
 // Crystallography.
//
 // Authors: Stefano Trapani and Jorge Navasa.
//
 //
//
 // Refer to macro wig_graph.C in tutorials directory for an example
//
 // how to use this class to sample wigner d functions.
//
 //
//
 ////////////////////////////////////////////////////////////////////////

//----------------------------------------------------------------------
TWignerd::TWignerd(Int_t lbandlim): fLBandLim(lbandlim+1), fL(0),
fSize(fLBandLim*(fLBandLim+1)/2 + fLBandLim),fMatrix(fSize)
{
   // lbandlim: Band limit of the signal to be analysed.

   fMatrix[fIndex(0,0)] = 1;
}

//----------------------------------------------------------------------
void TWignerd::Recurse()
{
   // Advances one step in the recursion, from l to l+1

   ++fL;

   if ( fL >= fLBandLim ){
      std::cerr << "Recurse, Bigger band limit required: l =" << fL << std::endl;
      return;
   }

   TArrayD fTemp(fSize);
```

```
   fTemp[fIndex(fL,0)] = -Sqrt((2*fL-1)/(2*fL))*fMatrix[fIndex(fL-1,0)];

   for (Int_t n = 1; n <= fL; ++n){
      Double_t a = Sqrt(((fL/2)*(2*fL-1))/((fL+n)*(fL+n-1)));
      fTemp[fIndex(fL,n)] = a*fMatrix[fIndex(fL-1,n-1)];
   }

   for(Int_t m=fL-1;m>=0;--m){
      for(Int_t n=0;n<=m;++n){
         Double_t a = static_cast<Double_t>(n << 1)/Sqrt((fL-m)*(fL+m+1));
         Double_t b = -Sqrt((fL-m-1)*(fL+m+2)/((fL-m)*(fL+m+1)));
         fTemp[fIndex(m,n)] = a*fTemp[fIndex(m+1,n)] + b*fTemp[fIndex(m+2,n)];
         }
   }

   fMatrix = fTemp;
}

//-----------------------------------------------------------------------
complex<Double_t> TWignerd::operator()(Int_t m, Int_t n, Int_t u) const
{
   // Returns Fourier coefficients of the wigner d-function d^l_mn.
   // If module of m or n or u greater than l returns zero instead of
   // reporting an error.
   // u: frequency

   complex<Double_t> c(0,0);

   if(Abs(u) <= fL && Abs(m) <= fL && Abs(n) <= fL){
      complex<Double_t> I(0,1);
      complex<Double_t> b(pow(I,n-m));
      c = b*Delta(u,m)*Delta(u,n);
   }

   return c;
}

//-----------------------------------------------------------------------
void TWignerd::Advance(Int_t l)
{
   // Advances using a loop in Recurse(), so that fL = l.

   for (Int_t i = 0; i < l; ++i)
      Recurse();
}

//-----------------------------------------------------------------------
Double_t TWignerd::Delta(Int_t m, Int_t n) const
{
   // Returns delta using symmetry relations.

   if (m < 0) {
      if (n < 0) {
         if (m >= n)
            return fMatrix[fIndex(-n,-m)];
         return Even(n-m) ? fMatrix[fIndex(-m,-n)]: -fMatrix[fIndex(-m,-n)];
```

```
      } else {
         if (n <= Abs(m))
            return Even(fL-n) ? fMatrix[fIndex(-m,n)]: -fMatrix[fIndex(-m,n)];
         return Even(fL-m) ? fMatrix[fIndex(n,-m)]: -fMatrix[fIndex(n,-m)];
      }
   }
   if (n < 0) {
      if (m >= Abs(n))
         return Even(fL-m) ? fMatrix[fIndex(m,-n)]: -fMatrix[fIndex(m,-n)];
      return Even(fL-n) ? fMatrix[fIndex(-n,m)]: -fMatrix[fIndex(-n,m)];
   } else {
      if (m >= n)
         return fMatrix[fIndex(m,n)];
      return Even(n-m) ? fMatrix[fIndex(n,m)]: -fMatrix[fIndex(n,m)];
   }
}
```