TemplateOverlap

Short tutorial

Jose Juknevich

Department of Particle Physics and Astrophysics
Rehovot, Israel 76100
jose.juknevich@weizmann.ac.il

November 5, 2012

TemplateOverlap

tom.hepforge.org

Goal

In this tutorial you will learn how to:

- Use the `TemplateOverlap` function `matchTemplate` to search for matches between a template and a input event
- Use the `TemplateOverlap` function `maximize` to find the maximum overlaps (as well as their template directions) in a given array.

Getting TemplateOverlap

`TemplateOverlap` is now available on Hepforge
Here is how to install it on a Linux/Unix/MacOSX system as a standalone package.

1. Download the (current) source tarball and compile it

```
wget http://www.hepforge.org/archive/tom/TemplateOverlap-x.y.z.tar.gz
tar xvf TemplateOverlap-x.y.z.tar.gz
cd TemplateOverlap-x-y-z
make
```

   At the time of writing, x.y.z=0.2.0.

2. This compiles the `example` program that illustrate several aspects of `TemplateOverlap`.

Types of input

- Text (ASCII)
  - Simple list of four-vectors
- Ntuples
  - Provided by Atlas group at?
- Any other Ntuple or text input that one writes a class for

Overview

Now that you have seen the `example` application (and perhaps even compiled and run it), you might be wondering how it works. The `example` application consists of three primary components:

- source code comments,
- the `MatchingMethod` class definition,
- and the main method.

The following explanation will provide you with a basic understanding of the code, but the deeper implications will only become apparent after you have finished reading the rest of the tutorial.

Theory

What is Template Overlap?

Template Overlap is a technique for finding regions in the energy flow of observed jets that match (are similar to) the flow of a selected template configuration.

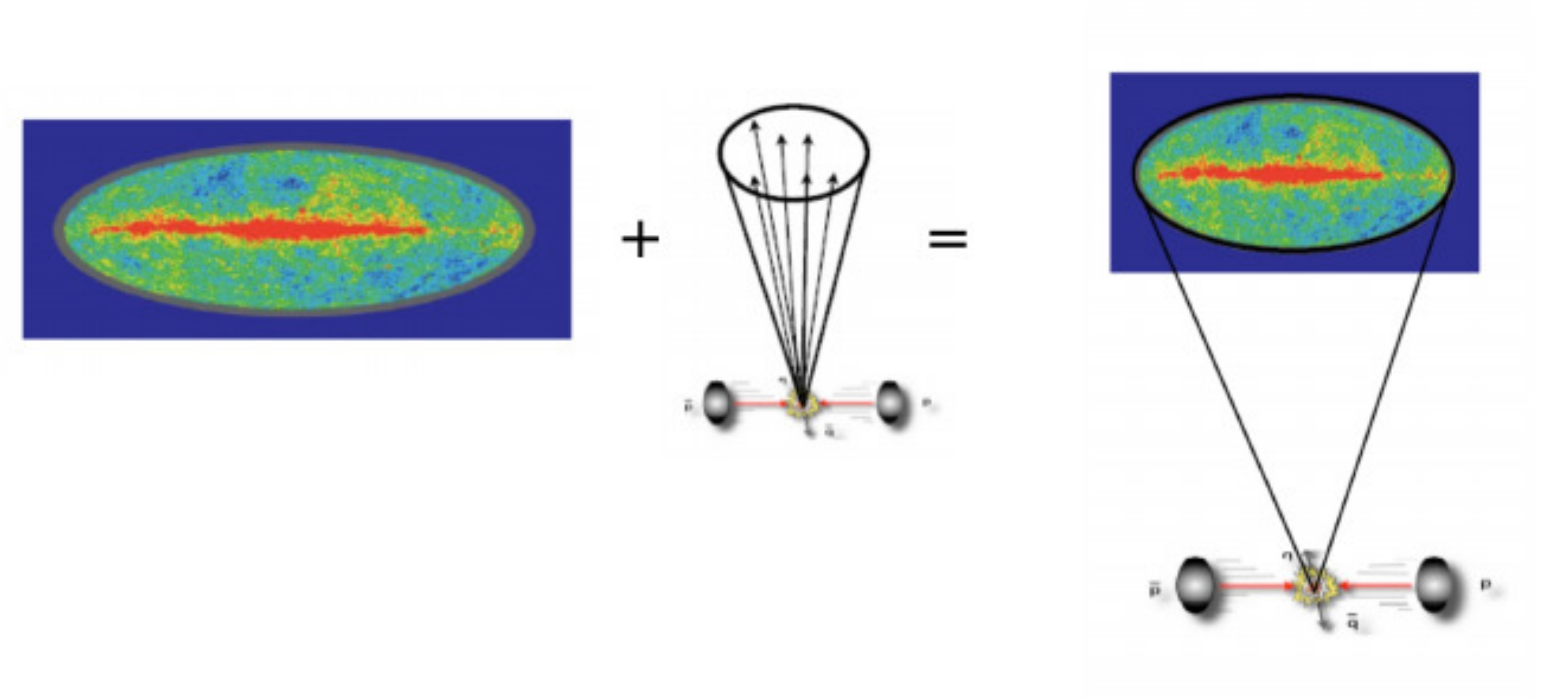
How does it work?

We need two primary components:

- **Source event (j):** The event in which we expect to find a match to the template event
- **Template (f):** The partonic configuration which will be compared to the source event.

Pattern recognition in energy flow

Our goal is to detect the highest matching events. That tells us about the likelihood that the event is signal or background:

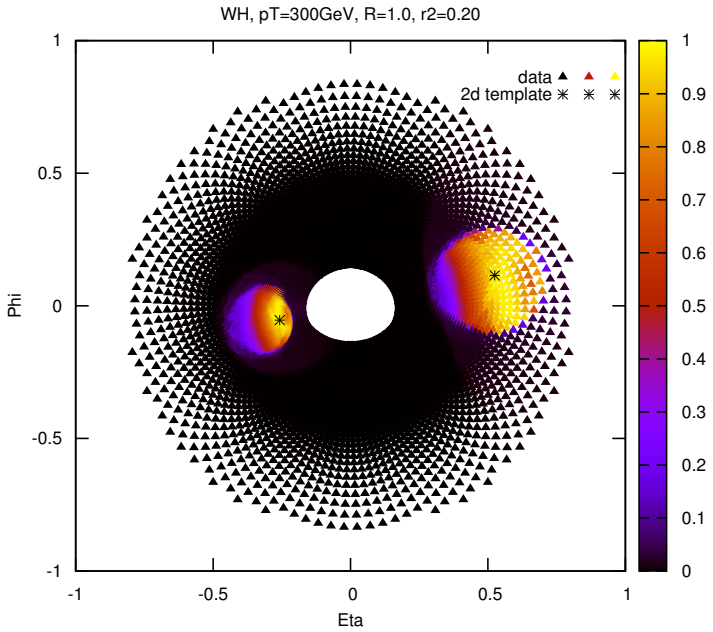


Template matching

- We compare the event against each template in the space of configurations, for every possible orientation of their decay products
- For each template, a measure is calculated so it represents how "good" or "bad" the match at the template is
- For each location of f over j, you store the measure in the result matrix $\mathbf{R}$.

Example: 2-particle templates

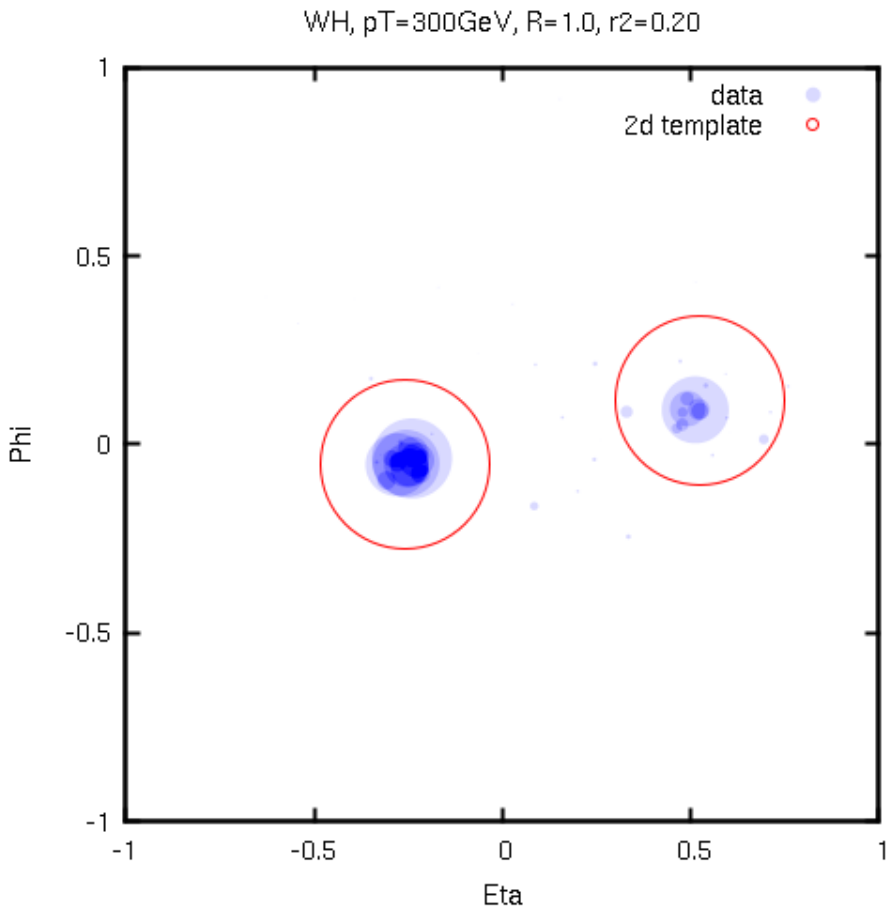
Each location (η, ϕ) in **R** contains the overlap measure:



The image above is the result **R** of running over templates with a Gaussian measure. The yellow-ish locations indicate the highest matches. The location marked by the black stars is probably the one with the highest value, so that location (the circle formed by that point as a corner and radius equal to the subcone of the template) is considered the match.

Maximize

In practise we use the function `maximize` to locate the highest value in the **R** matrix.



What are the matching methods available?

`TemplateOverlap` implements template matching in the function `matchTemplate`. The available methods are two:

1. **mode=Cone**

$$F(\Omega, f) = \max\left[1 - \frac{\eta^2 + \phi^2}{R^2}, 0\right] \tag{1}$$

2. **mode=Gaussian**

$$F(\Omega, f) = \exp\left[-(\eta^2 + \phi^2)/(2\sigma^2)\right], \tag{2}$$

The parameter σ determines the radial scale of the subjet reconstruction, analogous to the subcone radius R.

Public code

What does this program do?

- Loads an input event and a query configuration (template)
- Perform a template matching procedure by using the `TemplateOverlap` function `matchTemplate` with either of the two matching methods described before.
- Normalize the output of the matching procedure so that unity means perfect match.
- Localize the template with higher matching probability
- Return the highest probability (overlap) and best matched configuration (maximum overlap template).

Download code here

`http://www.hepforge.org/archive/tom/TemplateOverlap-0.2.0.tar.gz`

Example code

```
int main(int argc, char* argv[]) {

  if (argc !=4) {
    std::cout << "Usage: "<< argv[0]
      << " <input.dat> <template.dat> <output.dat>" << std::endl;
    return 1;
  }
   /// Set Run Parameters
 double subConeRadius = 0.20;
 double coneRadius = 1.0;
 double sigma = 0.333 ;
 settings myParms(subConeRadius, coneRadius, sigma);

 /// Create the jet and best matched template containers
 Event myJet, maxTempl;
 double maxOv;

/// Create an instance of Cone for the analysis.
  MatchingMethod myCone(argv[3], myParms);

 /// Load jet and templates
  myCone.templateRead(argv[2]);
  myJet.eventRead(argv[1]);

 ///  Do the matchig and localize the best match
  myCone.analyze(myJet, maxTempl, maxOv);

return 0;
}
```

Example code

```
void MatchingMethod::analyze (const Event myJet, Event & maxTempl, double & maxOv)
{
  /// Defining TemplateOverlap parameters
  int match_method = CONE;
  int normalize = HADRONIC;

   /// Source jet to matrix
  jet_t jet_mat;
  myJet.copyTo(jet_mat);

  /// Create the result matrix
  vector<double>  result;

  /// Do the matching and normalize
  matchTemplate(jet_mat, _templates, result, match_method, normalize);

  /// Localizing the best match with minMaxLoc
  double maxVal; int maxLoc;
  maximize(result, maxVal, maxLoc );

  /// Result to Event
  maxTempl.getFrom(_templates[maxLoc]);
  maxOv = maxVal;

  /// Print data to an output file
  _mystream << maxVal << " " << maxLoc << std::endl;
}
```

What's next?

- finish Gaussian kernel implementation
- add grid discretization for data preclustering
- add `ROOT` imput classes
- more documentation

`TemplateOverlap` is in a perpetual state of alpha release—we always have our latest stable build on display. Along the way we do test-driven development, code reviews... Help us do it faster and better.

Please visit our website at `tom.hepforge.org`