SCETLIB.

Frank Tackmann

Deutsches Elektronen-Synchrotron

SCET 2018 Workshop
Amsterdam, March 21, 2018

Main developers:
Markus Ebert, Johannes Michel, FT

So.

We have worked out our factorization theorem ... ✓

So.

We have worked out our factorization theorem ... ✓

We have calculated our matching, anomalous dimensions, ... ✓

So.

We have worked out our factorization theorem ... ✓

We have calculated our matching, anomalous dimensions, ... ✓

Now we actually want to get some numbers out ...

- Numerical implementation often time-consuming and nontrivial
  - Mathematica is very expedient for playing, exploring, quick’n’easy plotting, testing profiles, ...
  - But it is also very slow and hard to maintain, scale, share, interface

So.

We have worked out our factorization theorem ... ✓

We have calculated our matching, anomalous dimensions, ... ✓

Now we actually want to get some numbers out ...

- Numerical implementation often time-consuming and nontrivial
  - Mathematica is very expedient for playing, exploring, quick'n'easy plotting, testing profiles, ...
  - But it is also very slow and hard to maintain, scale, share, interface

Eventually, we might also want others (theorists and experimentalists alike) to be able to use our results ...

- Nontrivial effort is required to go from a working code we can use ourselves to a code our collaborators can use
- Nontrivial effort is required to go from a code our collaborators can use to a code everyone can use

Design and Philosophy.

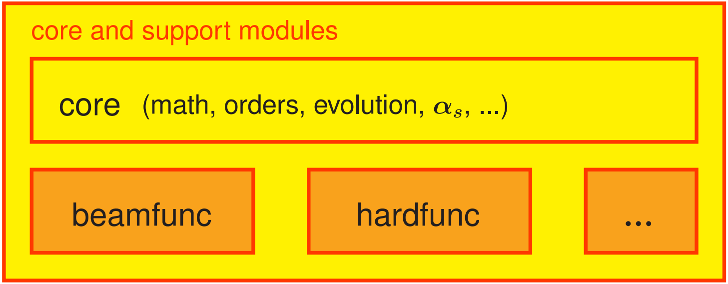
In contrast to existing resummation codes, SCETLIB is meant and designed as a *library* (mostly written in C++)

- Not only a black box to (re)produce an existing result but also a toolbox to build new results
  - Barrier of adoption for new result is significantly lowered when available in an already familiar format/framework

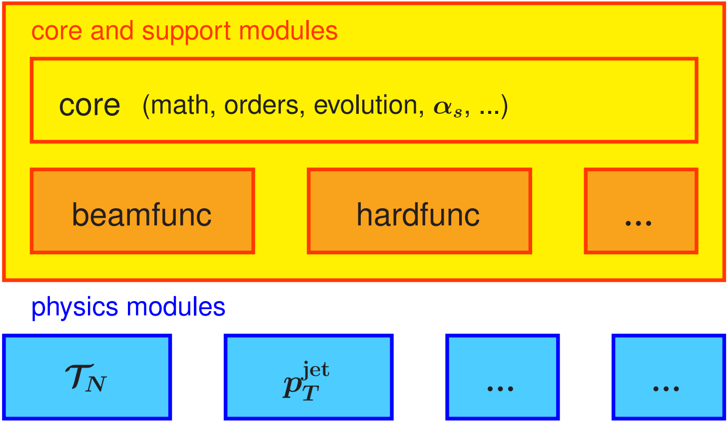
Main design goals

- User-friendly
  - Intuitive (physics-driven) and powerful library interface (API) for toolbox users
  - Ease of use for end (blackbox) users
  - Safety against unintentional or accidental misuse for either
- Modular design
  - Reuse and rely on existing, validated, well-tested components
  - Infrastructure to assemble building blocks
  - Allow for flexibility, extendability, scaleability
- Stability and speed

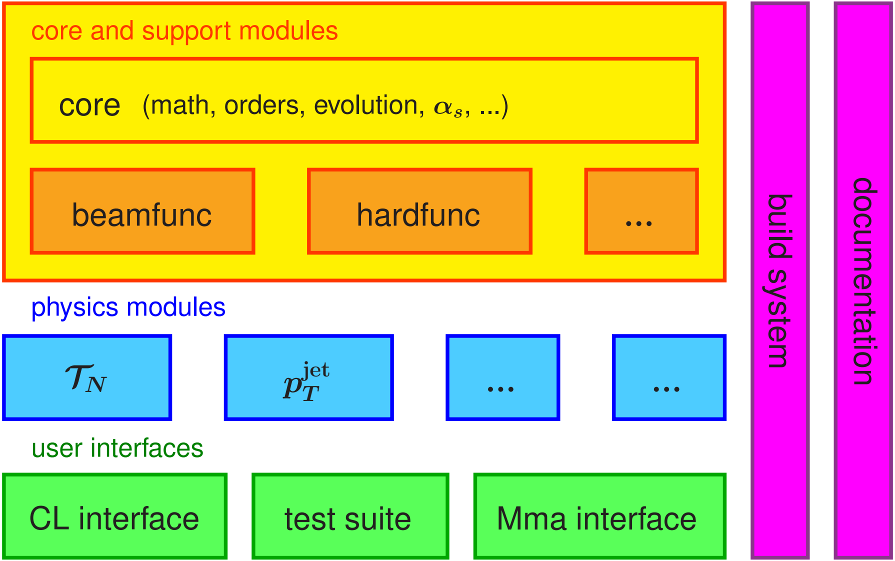
Overview of SCETLIB.



Overview of SCETLIB.



Overview of SCETLIB.



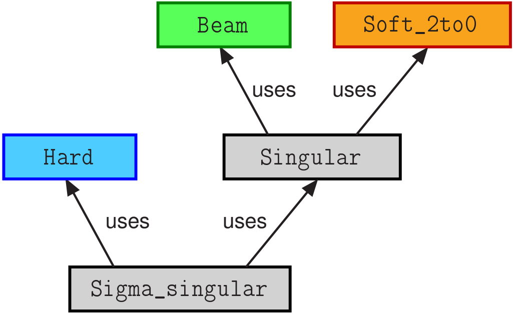
Factorization as Design Principle.

“Factorization” is one of the core design principles in C++ (and in general)

- Break up problem into smaller pieces
  - Each piece does one thing and does it well
  - Complexity is achieved through their interactions
- For us, it means that the logical (i.e. physics) factorization is directly reflected in the code
- Example:

  Structure of $\mathcal{T}_N$ module

$$\sigma(Q, Y, \tau) = H(Q) \times \left[B(Qe^{Y}) \otimes B(Qe^{-Y}) \otimes S\right](\tau)$$



Example: Interface of TauN::Singular.

```
class Singular
{
   public:
      using Phi = Phi_label_2to0;
      using Color = Soft_2to0::Color;

      // constructor
      Singular(Color color, Resum_order order, RunningCoupling<>& alphas,

      // Returns distribution-valued perturbative series.
      JointDistribution operator()(Phi phi, Scales scales) const;

      // Evaluate the spectrum at Tau.
      auto spectrum(Phi phi, double Tau, Scales scales) const;

      // Evaluate the cumulant up to TauCut.
      auto cumulant(Phi phi, double TauCut, Scales scales) const;

   private:
      Resum_order _order;                // resummation order
      RunningCoupling<>& _alphas;        // running coupling
      Beam _beam;                        // beam functions
      Soft_2to0 _soft;                   // soft function
};
```

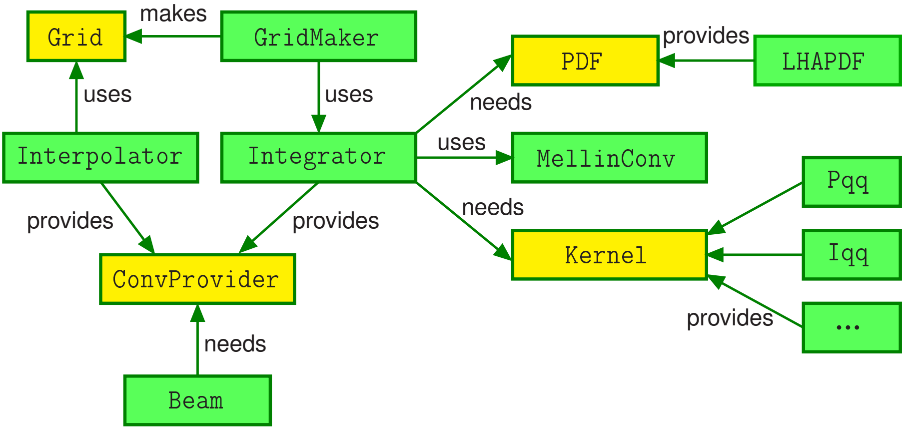
Example: Implementation of TauN::Singular.

```
// constructor
Singular::Singular(Color color, Resum_order order,
                   RunningCoupling<>& alphas, ...)
   : _order(order), _alphas(alphas),
     _beam(color == qqbar ? Beam::quark : Beam::gluon, order, alphas, ...)
     _soft(color, order, alphas)
{}

// _beam and _soft return distribution-valued perturbative series
// including RG evolution.
// Their multiplication operator* evaluates their convolution.
JointDistribution Singular::operator()(Phi phi, Scales scales) const
{
   return _beam(phi.channel.fa, phi.wa, scales.muBa, scales.mu)
          * _beam(phi.channel.fb, phi.wb, scales.muBb, scales.mu)
          * _soft(scales.muS, scales.mu);
}

// Evaluate the spectrum at Tau.
auto spectrum(Phi phi, double Tau, Scales scales) const
{
   return (*this)(phi, scales).spectrum(Tau);
}
```

Beam Function Module.

Facilities for beam function coefficients and convolutions $\int \frac{dz}{z}\, I_{ij}(z)\, f_j\!\left(\frac{x}{z}\right)$

- Flexible: Different ways to provide convolutions
  (via fast grid interpolation and/or on-the-fly integration)
- Extendable: Easy to add new kernels (or other provider strategies)



Mathematica Interface.

Mathematica is convenient for plotting etc., but interfacing it to external C++ code can be excruciating (if you ever tried, you know what I mean ...)

SCETLIB's Mathematica interface provides access to its functionality from Mathematica in a few easy steps

- Write a simple class that exposes the desired functionality

```
//MMA_EXPORT_CLASS//
class DrellYan
{
   public:
      //MMA_EXPORT//
      DrellYan(...) { ... } // setup the _sigma

      //MMA_EXPORT//
      double spectrumResummedQY(double Q, double Y, double Tau,
                                complex muH, double muB, double muS)
      {
         return _sigma.spectrum(Phi{Q, Y}, Tau, muH, Scales{muB, muS, Q});
      }

   private:
      Sigma_singular<hardfunc::DrellYan> _sigma;
}
```

Mathematica Interface.

Mathematica is convenient for plotting etc., but interfacing it to external C++ code can be excruciating (if you ever tried, you know what I mean ...)

SCETLIB's Mathematica interface provides access to its functionality from Mathematica in a few easy steps

- Write a simple class that exposes the desired functionality
- Run the MmaInterface, which automatically
  - Parses the class definitions
  - Generates source code for a Wolfram LibraryLink library and builds it
  - Generate a corresponding Mma package to use the LibraryLink library
  - Can also export multiple classes in one package
- In Mathematica
  - Load the package
  - Create objects of the exported class(es), where each object can have its own settings and multiple objects can coexist
  - Call their exported member functions
  - Errors (exceptions) in the C++ code are caught and passed through as Mma warnings

→ Live Demo ...

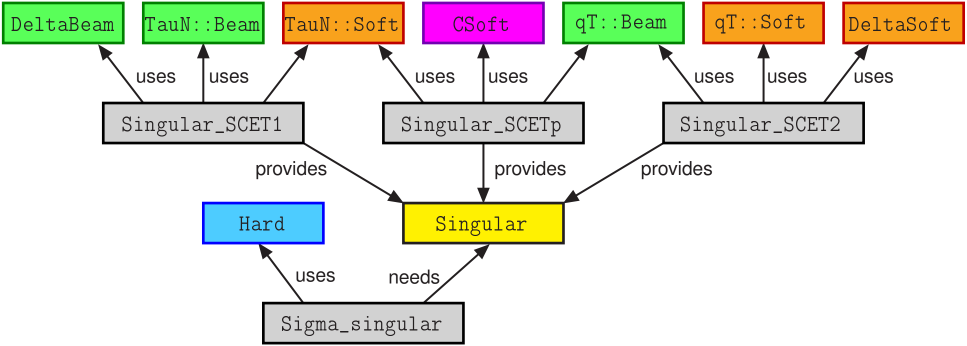
Nontrivial Example: Double-differential $\mathcal{T}_0 - q_T$

[→ see Gillian's talk for the physics details]

$$\sigma_{\mathrm{I}}(\mathcal{T}_0, q_T) = H \times B(\mathcal{T}_0, q_T)^2 \otimes S(\mathcal{T}_0)$$
$$= B(\mathcal{T}_0) + \Delta B(\mathcal{T}_0, q_T)$$

$$\sigma_{+}(\mathcal{T}_0, q_T) = H \times B(q_T)^2 \otimes \mathcal{S}(\mathcal{T}_0, q_T)^2 \otimes S(\mathcal{T}_0)$$

$$\sigma_{\mathrm{II}}(\mathcal{T}_0, q_T) = H \times B(q_T)^2 \otimes S(\mathcal{T}_0, q_T)$$
$$= S(q_T) + \Delta S(\mathcal{T}_0, q_T)$$



Summary and Outlook.

SCETLIB strives to be an easy-to-use, powerful, multi-purpose library

- Significantly reduce effort to numerically implement new calculations
- Rely on tested and validated implementations of existing ingredients
- Make results available to the experimental and theoretical community

⇒ Get more easily

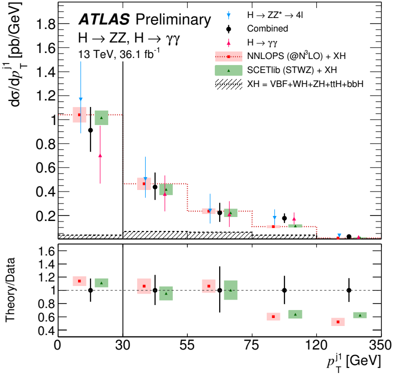
from here

$$
\begin{aligned}
\sigma_0(p_T^{\rm cut}) &= \sigma_B H_{gg}(m_t, m_H, \mu_H) \int \mathrm{d}Y\, B_g(m_H, p_T^{\rm cut}, R, x_a, \mu_B, \nu_B) \\
&\quad \times B_g(m_H, p_T^{\rm cut}, R, x_b, \mu_B, \nu_B)\, S_{gg}(p_T^{\rm cut}, R, \mu_S, \nu_S) \\
&\quad \times U_0(p_T^{\rm cut}, R; \mu_H, \mu_B, \mu_S, \nu_B, \nu_S) \\
&\quad + \sigma_0^{R\mathrm{sub}}(p_T^{\rm cut}, R) + \sigma_0^{\rm ns}(p_T^{\rm cut}, R, \mu_{\rm ns})\,, \qquad (56)
\end{aligned}
$$

where the combined renormalization group evolution factor U_0 is given by

$$
\begin{aligned}
U_0(p_T^{\rm cut}, R; \mu_H, \mu_B, \mu_S, \nu_B, \nu_S) &= \left| \exp\left[\int_{\mu_H}^{\mu_B} \frac{\mathrm{d}\mu'}{\mu'}\, \gamma_H^g(m_H, \mu') \right] \right|^2 \\
&\quad \times \exp\left[\int_{\mu_S}^{\mu_B} \frac{\mathrm{d}\mu'}{\mu'}\, \gamma_S^g(\mu', \nu_S) \right] \\
&\quad \times \exp\left[\ln\frac{\nu_B}{\nu_S}\, \gamma_\nu^g(p_T^{\rm cut}, R, \mu_B) \right]. \qquad (57)
\end{aligned}
$$

to here



Summary and Outlook.

SCETLIB strives to be an easy-to-use, powerful, multi-purpose library

- Significantly reduce effort to numerically implement new calculations
- Rely on tested and validated implementations of existing ingredients
- Make results available to the experimental and theoretical community

Outlook

- Current v0.4: Not-yet public
  - Could only show you some of the features
- Next v0.5: Will be public
  - If interested, watch this space: **http://scetlib.desy.de**
  - or tell me to add you to the **scetlib-announce@desy.de** email list