P4Helpers

P4Helpers.h provides static helper functions for kinematic calculation on objects deriving from I4Momentum.

Anthony Denton-Farnworth,

The University of Sheffield

Aim

- To reduce the total processing time for a number of kinematic calculations between I4Momentum derived objects whilst maintaining a standardised interface for end users.

Basic Method

- When a relationship between two kinematic quantities is calculated, each of the quantities themselves may need to be calculated first.
- By combining all three calculations together in the same function some terms may be cancelled or simplified. The simplified form can then be processed more efficiently.

An Analogy

- Consider the calculation of a definite integral within some model of a system. In the simple case, with sufficient information on the boundary conditions relating to our system, we could:
  - calculate the value of the indefinite integral (including the constant of integration) at the start point
  - calculate the value of the indefinite integral (including the constant of integration) at the end point
  - Take the difference of the two values

An Analogy

- Of course, in practice this is not the method we would typically use. Knowing that the constant of integration will cancel at the end, we typically perform this cancellation implicitly.
- Yet, something similar to the method described may occur if we naively compare the members of I4Momentum objects directly – what is necessary for calculating an individual kinematic variable will often be less that optimal for calculating relationships between multiple variables.

An Example

- Taking ΔEta for example:
- The calculation of each Eta value may include a computationally expensive log term.
- Instead of:

  ΔEta = Eta1 – Eta2 = log(arg1) – log(arg2)

  We can use the faster:

  ΔEta = log(arg1/arg2)
- By this method we do not ever actually calculate the two Eta values themselves but we do calculate ΔEta efficiently – which is all we need.

I4Momentum Kinds

- Case-by-case optimisations are offered for various pairings of I4Momentum kinds:
  - P4EEtaPhiM
  - P4IPtCotThPhiM
  - P4PtEtaPhiM
  - P4PxPyPzE
- These kinds are based on intermediate classes in the I4Momentum inheritance hierarchy and, among other things, define which properties are stored and accessible directly.

I4Momentum Kinds

- The P4Helpers establish kind at run-time and so will always call the best implementation without the need for the user to explicitly type-check and cast down.
- This makes the helpers very flexible and allows for optimisation even when used within generalised functions that must serve multiple purposes and where assumptions about the I4Momentum kind cannot be made.

The Code

- The code is contained within the namespace: P4Helpers
- The functions can typically take const pointers or const references to two I4Momentum objects
- The functions are all declared inline.
- Also included are efficient sort functions and matching functions (e.g. finding the closest DeltaR between some I4Momentum and a collection of I4Momenta)

  (See end for a list of functions)

Performance Gains

- An initial version produced an improvement of 3% in the processing time for RDO to ESD with 1000 ttbar events.
- This improvement is expected to increase in future in two ways:
  - A more complete implementation of possible optimisations within P4Helpers itself
  - Wider use of P4Helper function calls in place of direct manipulation of I4Momentum members

Further Advantages

- Additional benefits of maintaining the code for these calculations centrally include:
  - Unnecessary duplication of effort can be avoided
  - The code will be exposed to wider use, so potential bugs may be noticed sooner than the same issues in an independent implementation
  - Any issues can be fixed quickly, and all users will benefit immediately from the fix
  - All users will profit immediately from future improvements to the optimisation

Progress

- Currently my work on this is just starting but there is already a good deal of functionality in place from other authors (see end).
- There is still good scope for further improvements as not all kinds have been optimised in the existing code.
- Where optimisations are not yet in place the helpers fall back automatically on existing functions

  —> Still possible to write the helpers into new code ready to benefit from the optimisation once released.

Future Possibilities

- One option for further improvement once the current work is complete is to offer similar functionality in a second package using overloaded functions.
- This would allow a programmer to call the best optimisation directly in circumstances where they have advance knowledge of the I4Momentum kind.

Function List

- The function list currently includes:
  - /// Computes efficiently \Delta{\eta}
    inline double deltaEta(const I4Momentum& p1,
    const I4Momentum& p2)
  - /// Computes efficiently \Delta{\eta}^2
    inline double deltaEtaSq(const I4Momentum& p1,
    const I4Momentum& p2)
  - /// Computes efficiently \Delta{\eta}, pointer args.
    inline double deltaEta(const I4Momentum * const p1,
    const I4Momentum * const p2)
  - /** delta Phi in range [-pi,pi[*/
    inline double deltaPhi(double phiA, double phiB)
  - /** delta Phi in range [-pi,pi[from one I4momentum reference */
    inline double deltaPhi(const I4Momentum& p4, const double phi)

Function List

- /** delta Phi in range [-pi,pi[from two I4momentum references */
  inline double deltaPhi(const I4Momentum & pA
  const I4Momentum & pB)
- /** delta Phi squared in range ([-pi,pi[)^2 from two I4momentum references */
  inline double deltaPhiSq(const I4Momentum & pA,
  const I4Momentum & pB)
- /** delta Phi in range [-pi,pi[from two I4momentum pointers */
  inline double deltaPhi(const I4Momentum * const pA,
  const I4Momentum * const pB)
- /** delta R from two I4momentum reference */
  inline double deltaR(const I4Momentum& pA,
  const I4Momentum& pB)
- /** delta R from two I4momentum pointers */
  inline double deltaR(const I4Momentum * const pA,
  const I4Momentum * const pB)

Function List

- /// Check if 2 @c I4Momentum are in a \Delta{R} cone
  /// @param dR [in] \Delta{R}
  /// @return true if they are
  inline bool isInDeltaR(const I4Momentum& p1,
  const I4Momentum& p2, double dR)
- /** invariant mass from two I4momentum references */
  inline double invMass(const I4Momentum & pA,
  const I4Momentum & pB)
- /** invariant mass from two I4momentum pointers */
  inline double invMass(const I4Momentum * const pA,
  const I4Momentum * const pB)
- /** invariant mass from four I4momentum references */
  inline double invMass(const I4Momentum & pA, const I4Momentum & pB, const I4Momentum & pC, const I4Momentum & pD)
- /** invariant mass from four I4momentum pointers */
  inline double invMass(const I4Momentum * const pA, const I4Momentum * const pB, const I4Momentum * const pC, const I4Momentum * const pD)

Function List

- /// sort a container according to the given predicate
  template<class Iterator_t, class Predicate_t>
  inline void sort(Iterator_t itr, Iterator_t itrEnd, Predicate_t p)
- /// sort a container according to the given predicate
  template<class Container_t, class Predicate_t>
  inline void sort(Container_t& container, Predicate_t p)
- /// Find the closest element in a collection to an @c I4Momentum
  /// @param index [out] index of the closest element
  /// @param deltaR [out] \Delta{R}
  /// @return true if found
  template <class Container_t>
  inline bool closestDeltaR(const double eta, const double phi,
  Container_t& coll, std::size_t& index, double& deltaR)

Function List

- /// Find the closest element in a collection to an @c I4Momentum
  /// @param index [out] index of the closest element
  /// @param deltaR [out] \Delta{R}
  /// @return true if found
  template <class Container_t>
  inline bool closestDeltaR(const I4Momentum& p4, Container_t& coll,
  std::size_t& index, double& deltaR)
- /// find the closest element in R - with a condition on E
  /// @param index [out] index of the closest element
  /// @param deltaR [out] \Delta{R}
  /// @param deltaE [out] \Delta{E}
  /// @return true if found
  template <class Container_t>
  inline bool closestDeltaR(const double eta, const double phi,
  const double ene, Container_t& coll, std::size_t& index,
  double &deltaR, double &deltaE)

Function List

- /// find the closest element in R - with a condition on E
  /// @param index [out] index of the closest element
  /// @param deltaR [out] \Delta{R}
  /// @param deltaE [out] \Delta{E}
  /// @return false if found
  template <class Container_t>
  inline bool closestDeltaR(const I4Momentum& p4, Container_t& coll,
  std::size_t& index, double &deltaR, double &deltaE)

Authors

- The authors of the original code are:
  - Sebastien Binet
  - Tadashi Maeno
  - David Rousseau
  - Rolf Seuster