Group meeting
May 3rd, 2024

Instruction responsibilities

- Classes:
  - PHY 252:
    - Finished writing final exam
    - Graded final exam
    - Posted grades
  - PHY 331:
    - Finished writing final exam
    - Graded final exam
    - Posted grades

- Undergraduate independent study and research:
  - No current undergrads
  - Potential new undergrad: Dylan Loew-Garrelts

Service responsibilities

- Committee:
  - GlueX Compton Analysis Review Committee:
    - Waiting for author response
  - Pre Award Faculty Panel:
    - Status : Done ☺

Group responsibilities

- Met with Katelyn Tuesday
- Sent (another) email to Lee Pettit asking if the paperwork for the summer hires (Katelyn and Alan) is completed. Was informed that all paperwork is in order ☺

Analysis

- Working on passing additional information from trees to PWA

Analysis

- Working on passing additional information from trees to PWA

Analysis

- Working on passing additional information from trees to PWA

What I want:

Analysis

- Working on passing additional information from trees to PWA

What I want:

- To include the polarization information (angle and degree) in the flattened tree (instead of breaking files into separate polarization types)

Analysis

- Working on passing additional information from trees to PWA

What I want:

- To include the polarization information (angle and degree) in the flattened tree (instead of breaking files into separate polarization types)

- Access the new tree information inside our AmpTools PWA calculation

Redefining our calcAmplitude function

Redefining our calcAmplitude function

In $AMPTOOLS/IUAmpTools/Amplitude.h

Redefining our calcAmplitude function

In $AMPTOOLS/IUAmpTools/Amplitude.h

```
/**
 * This is the user-defined function that computes a single complex amplitude
 * for a set of four-vectos that describe the event kinematics.  As discussed
 * above this function should be factorized as much as possible and not
 * include permutations of particles.  The user must override this function
 * in his or her amplitude class.
 *
 * \param[in] pKin a pointer to a single event.  pKin[0][0-3] define E, px,
 * py, pz for the first particle, pKin[1][0-3] for the second, and so on
 *
 */
virtual complex< GDouble > calcAmplitude( GDouble** pKin ) const ;
```

Redefining our calcAmplitude function

In $AMPTOOLS/IUAmpTools/Amplitude.h

```
/**
 * This is the user-defined function that computes a single complex amplitude
 * for a set of four-vectos that describe the event kinematics.  As discussed
 * above this function should be factorized as much as possible and not
 * include permutations of particles.  The user must override this function
 * in his or her amplitude class.
 *
 * \param[in] pKin a pointer to a single event.  pKin[0][0-3] define E, px,
 * py, pz for the first particle, pKin[1][0-3] for the second, and so on
 *
 */
virtual complex< GDouble > calcAmplitude( GDouble** pKin ) const ;
```

What we currently use

Redefining our calcAmplitude function

What we currently use

```
virtual complex< GDouble > calcAmplitude( GDouble** pKin ) const ;
```

Redefining our calcAmplitude function

What we currently use

```
virtual complex< GDouble > calcAmplitude( GDouble** pKin ) const ;
```

```
/**
 * This is the user-defined function that computes a single complex amplitude
 * for a set of four-vectos that describe the event kinematics.
 *
 * For the user to utilize user-defiend data in the amplitude calculation,
 * this function must be overridden by the derived class.  Either this function
 * or the function above must be defined for any Amplitude class.
 *
 * \param[in] pKin a pointer to a single event.  pKin[0][0-3] define E, px,
 * py, pz for the first particle, pKin[1][0-3] for the second, and so on
 *
 * \param[in] userVars is an optional pointer to the user data block associated
 * with this event and this permutation of particles.  It can be used to store
 * intermediate portions of the calculation in the case that calcAmplitude
 * must be called multiple times during the course of a fit.  The userVars
 * memory block is filled in calcUserVars.
 */
virtual complex< GDouble > calcAmplitude( GDouble** pKin,
                                          GDouble* userVars ) const;
```

What I want to use

Redefining our calcAmplitude function

What we currently use

```
virtual complex< GDouble > calcAmplitude( GDouble** pKin ) const ;
```

```
/**
 * This is the user-defined function that computes a single complex amplitude
 * for a set of four-vectos that describe the event kinematics.
 *
 * For the user to utilize user-defiend data in the amplitude calculation,
 * this function must be overridden by the derived class.  Either this function
 * or the function above must be defined for any Amplitude class.
 *
 * \param[in] pKin a pointer to a single event.  pKin[0][0-3] define E, px,
 * py, pz for the first particle, pKin[1][0-3] for the second, and so on
 *
 * \param[in] userVars is an optional pointer to the user data block associated
 * with this event and this permutation of particles.  It can be used to store
 * intermediate portions of the calculation in the case that calcAmplitude
 * must be called multiple times during the course of a fit.  The userVars
 * memory block is filled in calcUserVars.
 */
virtual complex< GDouble > calcAmplitude( GDouble** pKin,
                                          GDouble* userVars ) const;
```

What I want to use

Redefining our calcAmplitude function

What we currently use

```
virtual complex< GDouble > calcAmplitude( GDouble** pKin ) const ;
```

```
/**
 * This is the user-defined function that computes a single complex amplitude
 * for a set of four-vectos that describe the event kinematics.
 *
 * For the user to utilize user-defiend data in the amplitude calculation,
 * this function must be overridden by the derived class.  Either this function
 * or the function above must be defined for any Amplitude class.
 *
 * \param[in] pKin a pointer to a single event.  pKin[0][0-3] define E, px,
 * py, pz for the first particle, pKin[1][0-3] for the second, and so on
 *
 * \param[in] userVars is an optional pointer to the user data block associated
 * with this event and this permutation of particles.  It can be used to store
 * intermediate portions of the calculation in the case that calcAmplitude
 * must be called multiple times during the course of a fit.  The userVars
 * memory block is filled in calcUserVars.
 */
virtual complex< GDouble > calcAmplitude( GDouble** pKin,
                                          GDouble* userVars ) const;
```

What I want to use

Additional things to think about

```
/**
 * If the user intendends to store intermediate calculations that are
 * static but associated with each event and each permutation of this
 * particles, then this method should be overriden with a function that
 * returns the number of user variables that will be stored.  It is
 * recommended these are indexed with an enum.  The user must also define
 * the calcUserVars method.
 */
virtual unsigned int numUserVars() const { return 0; }
```

Additional things to think about

```
/**
 * If the user intendends to store intermediate calculations that are
 * static but associated with each event and each permutation of this
 * particles, then this method should be overriden with a function that
 * returns the number of user variables that will be stored.  It is
 * recommended these are indexed with an enum.  The user must also define
 * the calcUserVars method.
 */
virtual unsigned int numUserVars() const { return 0; }
```

Additional things to think about

```
/**
 * If the user intendends to store intermediate calculations that are
 * static but associated with each event and each permutation of this
 * particles, then this method should be overriden with a function that
 * returns the number of user variables that will be stored.  It is
 * recommended these are indexed with an enum.  The user must also define
 * the calcUserVars method.
 */
virtual unsigned int numUserVars() const { return 0; }

/**
 * If the user can calculate the amplitude from only the user-computed
 * data block and there is no need for the four-vectors, then the
 * user should override this function and return true.  If all amplitudes
 * in a fit can be calculated from user data then the memory consumption
 * in GPU fits can be optimizes as the raw four-vectors will not
 * be copied to the GPU.
 */

virtual bool needsUserVarsOnly() const { return false; }
```

Wants/Problems/Solutions

Wants/Problems/Solutions

Wants:

- I want to use variables from a root tree to set userVars

Wants/Problems/Solutions

Wants:

- I want to use variables from a root tree to set userVars
- I want to access my userVars within my amplitude function

Wants/Problems/Solutions

Wants:

- I want to use variables from a root tree to set userVars
- I want to access my userVars within my amplitude function

Problems:

- I do not know how to do the above tasks ☹

Wants/Problems/Solutions

Wants:

- I want to use variables from a root tree to set userVars
- I want to access my userVars within my amplitude function

Problems:

- I do not know how to do the above tasks ☹

Potential solutions:

- Read code [in progress]

Wants/Problems/Solutions

Wants:

- I want to use variables from a root tree to set userVars
- I want to access my userVars within my amplitude function

Problems:

- I do not know how to do the above tasks ☹

Potential solutions:

- Read code [in progress]
- Contact Matt Shepherd [sent email this morning]

Wants/Problems/Solutions

Wants:

- I want to use variables from a root tree to set userVars
- I want to access my userVars within my amplitude function

Problems:

- I do not know how to do the above tasks ☹

Potential solutions:

- Read code [in progress]
- Contact Matt Shepherd [sent email this morning]
- Start testing code modifications [details next slide]

Code modifications

- Changed calcAmplitude to have 2 arguments:

```
complex< GDouble >
Amp_R::calcAmplitude( GDouble** pKin , GDouble* userVars) const {
```

Code modifications

- Changed calcAmplitude to have 2 arguments:

```
complex< GDouble >
Amp_R::calcAmplitude( GDouble** pKin , GDouble* userVars) const {
```

New

Code modifications

- Changed calcAmplitude to have 2 arguments:

```
complex< GDouble >
Amp_R::calcAmplitude( GDouble** pKin , GDouble* userVars) const {
```

- Code compiled ☺
- Code successfully ran ☺