

Numerical Object Oriented Quantum Field Theory Calculations

M. Williams

Carnegie Mellon University, Pittsburgh PA, 15213

Abstract

The `qft++` package is a library of C++ classes that facilitate numerical (not algebraic) quantum field theory calculations. Mathematical objects such as matrices, tensors, Dirac spinors, polarization and orbital angular momentum tensors, etc. are represented as C++ objects in `qft++`. The package permits construction of code which closely resembles quantum field theory expressions, allowing for quick and reliable calculations.

1. Introduction

It is often desirable to describe particle physics processes using a covariant tensor formalism. This formalism contains Dirac spinors and matrices, polarization and orbital angular momentum tensors, etc. For cases involving higher spin particles or large values of orbital angular momentum, the presence of high rank tensors can make explicit calculation of the desired quantities (typically scattering or decay amplitudes) cumbersome at best and impossible at worst.

A common approach is to perform algebraic manipulations to reduce the number of tensor contractions prior to coding the expressions of interest. There are several software packages available to facilitate this approach, *e.g.* `FeynCalc` [1] and `FeynArts` [2]. Packages like `GRACE` [3] and `CompHEP` [4] take this a step further by (mostly automatically) producing distributions from physical models. While these packages are extremely useful in certain areas of physics (*e.g.* writing event generators), they are not ideal for performing an event-based partial wave analysis (PWA).

In this type of analysis, numeric values for amplitudes must be calculated for upwards of 100 million events (data and/or Monte Carlo). Thus, the amplitude calculation software must not only be easy to use, but also be capable of performing numerical

calculations of expressions involving matrices and high-rank tensors using as little cpu time as possible. The `qft++` package satisfies both of these criteria.

The operations of matrix multiplication and tensor contraction (the building blocks of all such calculations) can be broken down into nested loops; thus, they are easy to perform (numerically) on a computer. The `qft++` package was developed to take advantage of this fact by performing numerical calculations (not algebraic manipulations), making these types of calculations easier and more accessible to a larger portion of the physics community. The `qft++` package has been used in a number of PWA's to date (see, *e.g.*, [5]) and has performed exceptionally well.

One further point before discussing specific details about the software, this package was designed to calculate tree-level expressions. No work has been done to provide a simple way of performing loop integrals.

2. Overview

The the main design goal for the application programming interface (API) was to make the code resemble the mathematical expressions as closely as possible. To facilitate this goal, each type of mathematical object (*e.g.* tensors, spinors, etc.) has a corresponding C++ object in `qft++`. Through the use of operator overloading, operations such as matrix

multiplication and tensor contraction are handled by the objects themselves; not by the user, *i.e.* the user never has to keep track of any indices.

As a very simple example, consider the electromagnetic vertex $e\bar{u}(p_1, m_1)\gamma^\mu u(p_2, m_2)\epsilon_\mu(p_\gamma, m_\gamma)$. After variable declaration and initialization (discussed below), this expression can be written using `qft++` as follows:

```
e*Bar(u1(m1))*gamma*u2(m2)*eps(mg);
```

The object types determine how “multiplication”, *i.e.* the `*` operator, is to be performed. The matrix multiplications and tensor contractions are handled internally by the objects. This feature allows for self-documenting code, greatly reducing the probability for mistakes.

3. Basic Operations

The two main types of operations required in quantum field theory calculations, but not contained in standard C++, involve matrices and tensors. From these constituents, it is easy to build classes which handle Dirac matrices, covariant projection operators, etc. This section describes how the `qft++` package implements these basic types of operations.

This package makes heavy use of template classes; thus, a number of template utilities have been developed for performance and/or API reasons. Among these are compile-time detection of inheritance and parameter passing optimization [6], along with selective inclusion of class methods [7].

Unlike many standard C++ template classes, all of the template classes defined in the `qft++` package are designed such that instantiations of different types are fully compatible, provided the types themselves have the necessary operators defined, *e.g.* the following code is legal:

```
SomeClass<T1> sc1;
SomeClass<T2> sc2;
sc1*sc2;
```

if the operator `*` is defined between types `T1` and `T2`.

3.1. Matrix Operations

Matrix operations are handled by the template class `Matrix`, which can store any data type which can be stored in a C++ STL container class, *i.e.* it can store any object which can be stored by `std::vector`. A complete set of methods are provided including those useful in Quantum Field theory calculations, such as `Trace` and `Adjoint`, along with all the necessary operators.

3.2. Tensor Operations

Tensor operations are handled by the template class `Tensor`, which can also store any data type which can be stored in a C++ STL container class; however, the most useful types are typically `double` and `complex<double>`. Methods are provided to perform Lorentz transformations, symmetrization and a number of other functions.

A number of operators are provided to perform tensor contractions. The `Tensor` class assumes that all indices are either raised or lowered. The field theory objects discussed in Section 4 use the former, *i.e.* they are contravariant. The `*` operator performs standard multiplication if either object is not a `Tensor` and contraction of a single index otherwise. For example, if `x` is a rank-2 `Tensor`, then the code snippet `3*x` simply multiplies each of the elements of `x` by three; however, the snippet `x*y` will evaluate the expression $x^{\mu\nu}y_\nu$ if `y` is also a rank-2 `Tensor`. Full contraction of all possible indices is performed using the `|` operator; thus, `x|y` evaluates the expression $x^{\mu\nu}y_{\mu\nu}$.

Symbolic Expression	qft++ Code
$x^\mu y_\mu$	<code>x*y</code>
$x^\mu y^\nu$	<code>x%y</code>
$x^{\mu_1\mu_2\ldots\mu_n}y_{\mu_1\mu_2\ldots\mu_n}$	<code>(x y)</code>

Table 1
Example tensor operations using the `Tensor` template class.

Additional classes are provided for the Minkowski metric, $g_{\mu\nu}$ (`MetricTensor`), and the Levi-Civita tensor, $\epsilon_{\mu\nu\alpha\beta}$ (`LeviCivitaTensor`). The `Vector4` template class, which is derived from `Tensor`, provides a number of additional methods specific to 4-vectors, *e.g.* `CosTheta` and `Beta`.

4. Object-Oriented Field Theory

From the `Tensor` and `Matrix` classes discussed above, all of the necessary objects for quantum field theory calculations can be constructed. In this section, a brief overview of the formalism will be presented followed by a discussion of the corresponding `qft++` class. More detailed descriptions of the formalism are given in [8,9]. Recent examples of applying this formalism to PWA - for which this code was developed - can be found in [5,10,11].

4.1. Integral Spin Wave Functions

The wave function of a particle with integral spin- J , 4-momentum p and spin projection to some quantization axis m , is described by a rank- J tensor, $\epsilon_{\mu_1 \dots \mu_J}(p, m)$. The Rarita-Schwinger conditions for integral spin- J are

$$p^{\mu_i} \epsilon_{\mu_1 \mu_2 \dots \mu_i \dots \mu_J}(p, m) = 0 \quad (1a)$$

$$\epsilon_{\mu_1 \dots \mu_i \dots \mu_j \dots \mu_J}(p, m) = \epsilon_{\mu_1 \dots \mu_j \dots \mu_i \dots \mu_J}(p, m) \quad (1b)$$

$$g^{\mu_i \mu_j} \epsilon_{\mu_1 \mu_2 \dots \mu_i \dots \mu_j \dots \mu_J}(p, m) = 0, \quad (1c)$$

for any μ_i, μ_j , and reduce the number of independent elements from 4^J to $(2J+1)$.

The spin- J projection operator, defined as

$$P_{\mu_1 \mu_2 \dots \mu_J \nu_1 \nu_2 \dots \nu_J}^{(J)}(p) = \sum_m \epsilon_{\mu_1 \mu_2 \dots \mu_J}(p, m) \epsilon_{\nu_1 \nu_2 \dots \nu_J}^*(p, m), \quad (2)$$

is used to construct the particle's propagator.

As a simple example, consider a massive spin-1 particle. The Rarita-Schwinger conditions for spin-1 simply require $p^\mu \epsilon_\mu(p, m) = 0$. Thus, in the particle's rest frame the energy component of the wave function is zero. The spatial components are then chosen to be

$$\vec{\epsilon}(\pm 1) = \mp \frac{1}{\sqrt{2}}(1, \pm i, 0), \quad \vec{\epsilon}(0) = (0, 0, 1). \quad (3)$$

The wave function can be obtained in any other frame through the use of Lorentz transformations. The spin-1 projection operator is given by

$$P_{\mu\nu}^{(1)}(p) = \sum_m \epsilon_\mu(p, m) \epsilon_\nu^*(p, m) = -g_{\mu\nu} + \frac{p_\mu p_\nu}{w^2}, \quad (4)$$

where w is the mass of the particle.

Wave functions for particles with integral spin are handled in `qft++` by the `PolVector` class. The

spin is set using the constructor, *e.g.* `PolVector eps(3)` would be used for a spin-3 particle. The `PolVector` object must then be initialized for a given 4-momentum. At this point, the user can decide whether or not the particle is to be on-shell. For example, if a spin-1 particle is initialized on-shell then the projection operator is given by

$$P_{\mu\nu}^{(1)}(p) = -g_{\mu\nu} + \frac{p_\mu p_\nu}{p^2}, \quad (5)$$

otherwise, it is calculated using (4). This option is also available in `qft++` for half-integral spin particles.

After initialization, the sub-states can be accessed via calls like `eps(m)`, which returns the `Tensor<complex<double>>` object for sub-state m . The spin- J projection operator can be accessed easily in the code using the method `eps.Projector()`, which returns an object of type `Tensor<complex<double>>` with a rank of $2J$.

4.2. Half-Integral Spin Wave Functions

The wave function for a spin-1/2 particle is described by a 4-component Dirac spinor, denoted as $u(p, m)$, where p and m again represent the 4-momentum and spin projection of the particle. The representation chosen here leads to the following form of the spinors:

$$u(p, m) = \sqrt{E+w} \begin{pmatrix} \chi(m) \\ \frac{\vec{\sigma} \cdot \vec{p}}{E+w} \chi(m) \end{pmatrix}, \quad (6)$$

where $E(w)$ is the energy(mass) of the particle and $\chi(m)$ are the standard non-relativistic 2-component spinors.

Wave functions for particles with higher half-integral spin, denoted as $u_{\mu_1 \dots \mu_{J-1/2}}(p, m)$, are constructed using tensor products of integral spin wave functions and the spin-1/2 spinors described above. The Rarita-Schwinger conditions for half-integral spin- J are [8]

$$(\gamma^\mu p_\mu - w) u_{\mu_1 \dots \mu_{J-1/2}}(p, m) = 0 \quad (7a)$$

$$u_{\mu_1 \dots \mu_i \dots \mu_j \dots \mu_{J-1/2}}(p, m) = u_{\mu_1 \dots \mu_j \dots \mu_i \dots \mu_{J-1/2}}(p, m) \quad (7b)$$

$$p^{\mu_i} u_{\mu_1 \dots \mu_i \dots \mu_{J-1/2}}(p, m) = 0 \quad (7c)$$

$$\gamma^{\mu_i} u_{\mu_1 \dots \mu_i \dots \mu_{J-1/2}}(p, m) = 0 \quad (7d)$$

qft++ Class	Symbol	Concept
Matrix<T>	a_{ij}	matrices of any dimension
Tensor<T>	$x_{\mu_1 \dots \mu_n}$	tensors of any rank
MetricTensor	$g_{\mu\nu}$	Minkowski metric
LeviCivitaTensor	$\epsilon_{\mu\nu\alpha\beta}$	totally anti-symmetric Levi-Civita tensor
DiracSpinor	$u_{\mu_1 \dots \mu_{J-1/2}}(p, m)$	half-integral spin wave functions
DiracAntiSpinor	$v(p, m)$	spin-1/2 anti-particle wave functions
DiracGamma	γ^μ	Dirac matrices
DiracGamma5	γ^5	
DiracSigma	$\sigma^{\mu\nu}$	
PolVector	$\epsilon_{\mu_1 \dots \mu_J}(p, m)$	integral spin wave functions
OrbitalTensor	$L_{\mu_1 \dots \mu_\ell}^{(\ell)}$	orbital angular momentum tensors

Table 2

A partial list of qft++ classes, a complete list can be found at [12].

$$g^{\mu_i \mu_j} u_{\mu_1 \dots \mu_i \dots \mu_j \dots \mu_{J-1/2}}(p, m) = 0, \quad (7e)$$

for any μ_i, μ_j , and reduce the number of independent elements from $4^{J+1/2}$ to $(2J+1)$.

The spin- J projection operator is then defined as

$$P_{\mu_1 \dots \mu_{J-1/2} \nu_1 \dots \nu_{J-1/2}}^{(J)}(p) = \frac{1}{2w} \times \sum_m u_{\mu_1 \dots \mu_{J-1/2}}(p, m) \bar{u}_{\nu_1 \dots \nu_{J-1/2}}(p, m). \quad (8)$$

For example, the spin-1/2 projection operator is simply $P^{(1/2)}(p) = \frac{1}{2w}(p^\mu \gamma_\mu + w)$, while for spin-3/2 the projection operator is given by

$$P_{\mu\nu}^{(\frac{3}{2})}(p) = -P^{(\frac{1}{2})}(p) \times \left(P_{\mu\nu}^{(1)}(p) + \frac{1}{3} P_{\mu\alpha}^{(1)}(p) \gamma^\alpha P_{\nu\beta}^{(1)}(p) \gamma^\beta \right). \quad (9)$$

Wave functions for particles with half-integral spin are handled in qft++ by the `DiracSpinor` class. The spin is set using the constructor, e.g. `DiracSpinor u(3/2.)` would be used for a spin-3/2 particle. As with the `PolVector` class, the `DiracSpinor` class must be initialized for a given 4-momentum. The sub-states can then be accessed via `u(m)`, which returns the `Matrix<Tensor<complex<double>>>` object for sub-state m . The spin- J projection operators can be easily accessed using the method `u.Projector()` which returns a 4×4 `Matrix` of rank- $(2J-1)$ `Tensor<complex<double>>` objects. We also note here that the quantity $\bar{u}(p, m)$ is obtained using the function `Bar(u(m))`.

4.3. Dirac Matrices

Classes are also provided to handle the Dirac matrices, γ^μ (`DiracGamma`) and γ^5 (`DiracGamma5`), along with $\sigma^{\mu\nu} \equiv \frac{i}{2}[\gamma^\mu, \gamma^\nu]$ (`DiracSigma`). Each of these classes is derived from the common base class `Matrix<Tensor<complex<double>>>`. Thus, they inherit all of the necessary `Matrix` and `Tensor` operators.

4.4. Orbital Angular Momentum Tensors

Two particles, with 4-momenta p_a and p_b , can be coupled to a state of pure orbital angular momentum, ℓ , using the operators $L_{\mu_1 \mu_2 \dots \mu_\ell}^{(\ell)}$. The total and relative momenta are defined as $P = p_a + p_b$ and $p_{ab} = \frac{1}{2}(p_a - p_b)$ respectively. The orbital-angular-momentum operators are then built using the relative momentum and the spin- ℓ projection operator as follows:

$$L_{\mu_1 \mu_2 \dots \mu_\ell}^{(\ell)} \propto P_{\mu_1 \mu_2 \dots \mu_\ell \nu_1 \nu_2 \dots \nu_\ell}^{(\ell)}(P) p_{ab}^{\nu_1} p_{ab}^{\nu_2} \dots p_{ab}^{\nu_\ell}. \quad (10)$$

These operators satisfy the Rarita-Schwinger conditions

$$P^{\mu_i} L_{\mu_1 \mu_2 \dots \mu_i \dots \mu_\ell}^{(\ell)} = 0 \quad (11a)$$

$$L_{\mu_1 \mu_2 \dots \mu_i \dots \mu_j \dots \mu_\ell}^{(\ell)} = L_{\mu_1 \mu_2 \dots \mu_j \dots \mu_i \dots \mu_\ell}^{(\ell)} \quad (11b)$$

$$g^{\mu_i \mu_j} L_{\mu_1 \mu_2 \dots \mu_i \dots \mu_j \dots \mu_\ell}^{(\ell)} = 0, \quad (11c)$$

for any μ_i, μ_j , which insure that they have $(2\ell+1)$ independent elements.

In the `qft++` package, orbital-angular-momentum operators are handled by the `OrbitalTensor` class. This class inherits from `Tensor<double>`; thus, after construction it can be used just like any other tensor. Setting the tensor elements of $L_{\mu_1 \dots \mu_\ell}^{(\ell)}$ (`OrbitalTensor` object `orbL`) for 4-momenta p_a and p_b (`Vector4<double>` objects `pa` and `pb`) is done by simply calling `orbL.SetP4(pa,pb)`.

4.5. Additional Utilities

Functions are also provided to calculate useful quantities such as Clebsch-Gordon coefficients, Wigner D-functions, Breit-Wigner and Regge propagators, etc.

5. Example Applications

In this section, a few simple examples will be examined. The `qft++` package computes the values of expressions numerically; thus, the 4-momenta of the particles involved must be known. In an event-based partial wave analysis, these would be obtained from the experimental data and/or Monte Carlo events. In the case that one wants to calculate theoretical angular distributions, cross sections, polarization observables, etc., events with the desired kinematics must be generated as input for the `qft++` code. In the examples below, the assumption is made that one of these methods is employed. For the example plots shown in this section, the latter method was used.

5.1. $X(2^-) \rightarrow \omega K \rightarrow \pi^+ \pi^- \pi^0 K$

Consider the decay of a particle, X , with spin-parity $J^P = 2^-$ into an ω and K via F -wave. The invariant decay amplitude for this process is proportional to

$$\mathcal{A} \propto \epsilon_\mu^*(p_\omega, m_\omega) L^{(3)\mu\nu\alpha}(p_{\omega K}) \epsilon_{\nu\alpha}(P, M), \quad (12)$$

where $p_\omega, m_\omega(P, M)$ are the momentum and spin projection of the $\omega(X)$ and $p_{\omega K}$ is the relative momentum of the ωK system.

For this example, assume that the X is produced via $e^+ e^-$ annihilation resulting in population of only the $M = \pm 1$ sub-states. The decay distribution is then obtained by calculating the intensity

$$\mathcal{I} \propto \sum_{M=\pm 1} \sum_{m_\omega=\pm 1,0} |\mathcal{A}|^2. \quad (13)$$

To calculate this distribution using `qft++`, the necessary variables must first be declared:

```
PolVector epso; // omega
PolVector epsx(2); // X
OrbitalTensor orb3(3); // L^3
Tensor<complex<double>> amp;
Vector4<double> p4o,p4k,p4x;
```

For each point at which \mathcal{I} is to be calculated, the 4-momenta must be set and the polarization states initialized using calls to `SetP4`; however, if the kinematics are such that the X mass is constant, then `p4x` and `epsx` will only need to be initialized once.

In the code, a loop would then be performed over all values of $\cos \theta$ (the decay angle of the ω in the X rest frame) for which the decay intensity is to be calculated (or over events). At each point, the calculation would be performed as:

```
double intensity = 0.;
for(Spin m = -1; m <= 1; m++){
    for(Spin mo = -1; mo <= 1; mo++){
        amp = conj(epsx(mo))*orb3|epsx(m);
        intensity += norm(amp());
    }
}
```

In this way, the value of \mathcal{I} can be calculated at any number of points in $\cos \theta$ (or, for any number of events).

It is worth examining this more carefully. The tensor contractions in the code above are performed by first evaluating the `*` operator which contracts $\epsilon_\mu^*(p_\omega, m_\omega)$ into the first index of $L^{(3)\mu\nu\alpha}(p_{\omega K})$. The result is a rank-2 `Tensor` whose two indices are contracted via the `|` operator into both indices of $\epsilon_{\nu\alpha}(P, M)$. The result is a rank-0 `Tensor<complex<double>>` whose value is accessed via the `()` operator.

To check that the code is working for this simple example, the intensity in (13) can be calculated in the X rest frame by making a slight modification to the non-relativistic helicity formalism solution as follows:

$$\mathcal{I} \propto \sum_{M,\lambda} |f(E_\omega/w_\omega, \lambda) (301\lambda|2\lambda) d_{M\lambda}^2(\theta)|^2, \quad (14)$$

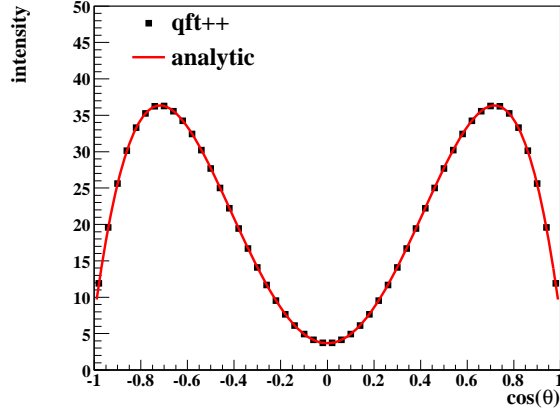


Fig. 1. (Color Online) Intensity vs $\cos\theta$: Angular distribution calculated for the decay $X(2^-) \rightarrow \omega K$ using **qft++** (black filled squares) compared with an analytic solution (red line). See text for details.

where $E_\omega, w_\omega, \lambda$ are the energy, mass and helicity of the ω and $f(x, \pm 1) = 1, f(x, 0) = x$ accounts for the effects of the boosts on the covariant ω helicity states. Notice that as $E_\omega \rightarrow w_\omega$, the non-relativistic solution is recovered.

The expression in (14) can then be rewritten purely in terms of the decay angle as follows:

$$\mathcal{I} \propto (2 \cos^2 \theta - 1)^2 + \cos^2 \theta + 9 \left(\frac{E_\omega}{w_\omega} \right)^2 \sin^2 \theta \cos^2 \theta. \quad (15)$$

Figure 1 shows the angular distributions obtained by calculating (12) using **qft++** compared to the modified helicity formalism expression given in (15), normalized to have the same integral as the **qft++** solution. The two calculations give the same angular distributions, *i.e.* the code is working properly.

This example can be extended by considering the secondary decay $\omega \rightarrow \pi^+ \pi^- \pi^0$. The amplitude for this process can be written as

$$\mathcal{A}_{\omega \rightarrow \pi^+ \pi^- \pi^0} \propto i \epsilon_{\mu\nu\alpha\beta} p_\pi^\nu p_\pi^\alpha p_{\pi^0}^\beta \epsilon^\mu(p_\omega, m_\omega), \quad (16)$$

which, in the ω rest frame, simplifies to

$$\mathcal{A}_{\omega \rightarrow \pi^+ \pi^- \pi^0}^{m_\omega} \propto i (\vec{p}_{\pi^+} \times \vec{p}_{\pi^-}) \cdot \vec{\epsilon}(m_\omega). \quad (17)$$

This is the standard non-relativistic result [13].

To incorporate this decay, (12) must be rewritten as

$$\mathcal{A} \propto \omega_\mu(p_\omega) L^{(3)\mu\nu\alpha}(p_{\omega K}) \epsilon_{\nu\alpha}(P, M), \quad (18)$$

where,

$$\omega_\mu(p_\omega) \propto \frac{i P^{(1)\mu\nu}(p_\omega)}{p_\omega^2 - w_\omega^2 + i w_\omega \Gamma_\omega} \epsilon_{\nu\rho\alpha\beta} p_\pi^\rho p_\pi^\alpha p_{\pi^0}^\beta, \quad (19)$$

and w_ω, Γ_ω are the mass and width of the ω .

To add this decay to the **qft++** calculation, the following variables need to be defined:

```
complex<double> i(0,1);
Tensor<complex<double>> omega;
LeviCivitaTensor levi;
Vector4<double> p4pip,p4pim,p4pi0;
```

Then, for each point the intensity is calculated as:

```
omega = epso.Projector()*levi
*p4pip*p4pim*p4pi0
*BreitWigner(p4o,0.78256,0.00844);
double intensity = 0.;
for(Spin m = -1; m <= 1; m+=2){
    amp = omega*orb3|epsx(m);
    intensity += norm(amp());
}
```

Since all of the objects are covariant, no extra boosts or rotations to the ω rest frame are required.

5.2. $\pi p \rightarrow \Delta \rightarrow \pi p$

As an example involving half-integral spin particles, consider the reaction $\pi p \rightarrow \Delta(1232) \rightarrow \pi p$. The invariant scattering amplitude for this process is proportional to

$$\mathcal{M} \propto \bar{u}(p_f, m_f) p_f^\mu P_{\mu\nu}^{(\frac{3}{2})}(P) p_i^\nu u(p_i, m_i), \quad (20)$$

where $p_i(p_f)$ and $m_i(m_f)$ are the 4-momentum and canonical spin projection of the initial(final) proton respectively. For this example, the mass-dependence of the propagator will be ignored. In the code, adding this would simply involve multiplying the amplitude by a **complex<double>**.

The distribution in the scattering angle, *i.e.* the angle between the initial and final protons ($\cos\theta = \hat{p}_i \cdot \hat{p}_f$), is obtained by calculating the scattering intensity

$$\mathcal{I} \propto \sum_{m_i, m_f} |\mathcal{M}|^2. \quad (21)$$

To calculate this distribution using `qft++`, the following variables must be declared:

```
DiracSpinor ui,uf; // proton spinors
DiracSpinor delta(3/2.);
Matrix<complex<double>> amp(1,1);
Vector4<double> p4_i,p4_f,p4_delta;
```

For each point at which \mathcal{I} is to be calculated, the 4-momenta must be set and the spinors initialized using calls to `SetP4`; however, if the 4-momentum of any given particle does change from point to point, then its corresponding object would only need to be initialized once.

In the code, a loop would then be performed over all values of $\cos\theta$ for which the scattering intensity is to be calculated (or, again, over all events). At each point, the calculation would be performed as:

```
double intensity = 0.;
for(Spin m_i = -1/2.; m_i <= 1/2.; m_i++){
  for(Spin m_f = -1/2.; m_f <= 1/2.; m_f++){
    amp = Bar(uf(m_f))*p4_f
          *delta.Projector()*p4_i*ui(m_i);
    intensity += norm(amp(0,0));
  }
}
```

In this way, the value of \mathcal{I} can be calculated at any number of points in $\cos\theta$.

If the $\Delta(1232)$ projector is on-shell (see discussion in Section 4.1), then the numerical calculations can be checked in the overall center-of-mass frame using the non-relativistic helicity formalism. In this frame, the scattering intensity is proportional to

$$\mathcal{I} \propto \sum_{\lambda_i, \lambda_f} |d_{\lambda_i \lambda_f}^{\frac{3}{2}}(\theta)|^2 \propto 1 + 3 \cos^2 \theta, \quad (22)$$

where $\lambda_i(\lambda_f)$ are the initial(final) proton helicities.

Figure 2 shows the angular distributions obtained by calculating (20) in the overall center-of-mass frame using `qft++` compared to the non-relativistic helicity formalism calculation used to obtain (22). The result obtained using (22) was normalized to have the same integral as the covariant calculation. Clearly the two calculations give the same angular distributions.

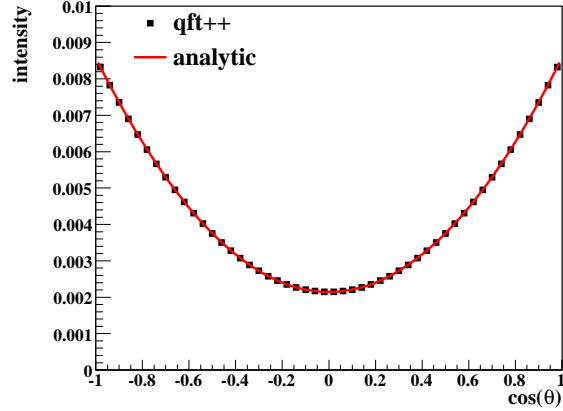


Fig. 2. (Color Online) Intensity vs $\cos\theta$: Angular distribution calculated for the reaction $\pi p \rightarrow \Delta(1232) \rightarrow \pi p$ using `qft++` (black filled squares) compared with an analytic solution (red line). See text for details.

5.3. Compton Scattering

As a final example, the unpolarized Compton scattering cross section will be calculated to leading order in α . The two well-known tree-level amplitudes, corresponding to s - and u -channel electron exchange diagrams, are

$$A_s = -e^2 \bar{u}(p', m'_p) \gamma^\mu \epsilon_\mu^*(k', m'_\gamma) \times \frac{\not{p} + \not{k} + w}{(p+k)^2 - w^2} \gamma^\nu \epsilon_\nu(k, m_\gamma) u(p, m_p) \quad (23a)$$

$$A_u = -e^2 \bar{u}(p', m'_p) \gamma^\nu \epsilon_\nu(k, m_\gamma) \times \frac{\not{p} - \not{k}' + w}{(p-k')^2 - w^2} \gamma^\mu \epsilon_\mu^*(k', m'_\gamma) u(p, m_p), \quad (23b)$$

where $p(p')$ and $k(k')$ denote the initial(final) electron and photon momenta respectively and w is the mass of the electron. The full scattering amplitude, from which the cross section can be calculated, is obtained by combining these two processes.

The calculation of this scattering amplitude is similar to that of the previous example. First, the following variables must be declared:

```

DiracSpinor ui,uf; // e spinors
DiracSpinor u_ex; // exchanged e spinor
PolVector epsi,epsf; // photon pol.vecs
DiracGamma gamma; // gamma^mu
// s- and u-channel propagators
Matrix<complex<double>> prop_s;
Matrix<complex<double>> prop_u;
Vector4<double> pi,pf,ki,kf; // 4-momenta

```

A loop would then be performed over all values of $\cos\theta$ (scattering angle in the lab frame) for which the cross section is to be calculated. At each point, the propagators are obtained using the following code:

```

u_ex.SetP4(pi+ki,0.511);
prop_s = u_ex.Propagator();
u_ex.SetP4(pi-kf,0.511);
prop_u = u_ex.Propagator();

```

The scattering intensity is then obtained by looping over spin projections, `Spin m_ef,m_ei,m_gf,m_gi`, and calculating the amplitudes given in (23) as

```

amp_s = Bar(uf(m_ef))*gamma
        *conj(epsf(m_gf))*prop_s*gamma
        *epsi(m_gi)*ui(m_ei);
amp_u = Bar(uf(m_ef))*gamma*epsi(m_gi)
        *prop_u*gamma*conj(epsf(m_gf))
        *ui(m_ei);

```

To get the cross sections the appropriate scale factors (α^2 , phase space factors, etc.) must then be applied.

Figure 3 shows the differential cross section for a 10 MeV incident photon calculated using `qft++` compared to the well-known spin-averaged Klein-Nishina formula [14]. Both methods give the same results.

6. Discussion

The examples in this paper were chosen because they can also easily be solved analytically, allowing for comparisons of the two calculations. In gen-

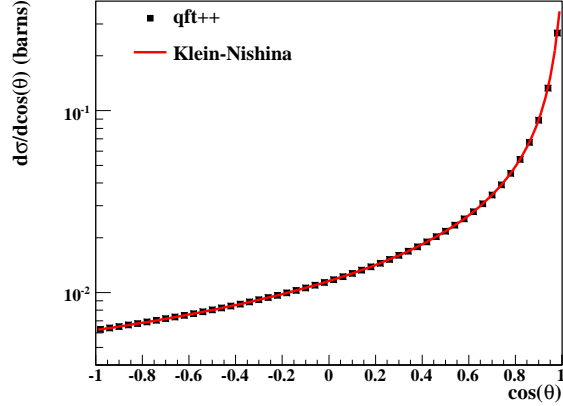


Fig. 3. (Color Online) $\frac{d\sigma}{d\cos\theta}$ (barns) vs $\cos\theta$: Compton scattering differential cross section for a 10 MeV incident photon calculated using `qft++` (black filled squares) compared with the Klein-Nishina equation (red line). See text for details.

eral, this is not the case; however, even for very complicated expressions involving high rank tensors or symmetrization, the `qft++` calculations are still very manageable (more complicated examples can be found online [12]). The code is also optimized for performance. The examples given in this paper run at ~ 3 kHz, *i.e.* on a 2.5 GHz processor, one can compute results at about 3000 kinematic points per second. This package makes it possible for any physicist to compare and/or fit a theoretical model to his or her data by coding up the expressions themselves.

It is clear from these examples how this package could be useful for a partial wave analysis, but it can also be used for other types of analyses. For example, it can be used to perform numeric checks of analytic calculations. The `qft++` package can also be used for cases where analytic solutions are desirable but not feasible. For example, consider the case where an expansion of an expression in powers of some variable, x , is needed but an analytic solution is not available. The `qft++` package could be used to evaluate the expression at a large number of values of x . The coefficients in the expansion could then be extracted by fitting the `qft++` results.

The `qft++` package is available to all users from <http://www-meg.phys.cmu.edu/qft++>.

Acknowledgments

I would like to thank Matt Shepherd for providing a more efficient version of the `TensorIndex`

class. This work was supported by grants from the United States Department of Energy No. DE-FG02-87ER40315 and the National Science Foundation No. 0653316 through the “Physics at the Information Frontier” program.

References

- [1] R. Mertig, M. Bhm and A. Denner. Comput. Phys. Commun. **64** 345 (1991).
- [2] T. Hahn. Comput. Phys. Commun. **140** 418 (2001).
- [3] J. Fujimoto *et al.* Comput. Phys. Commun. **153** 106 (2003).
- [4] E. Boos *et al.* Nucl.Instrum. Meth. **A534** 250 (2004).
- [5] M. Williams, Carnegie Mellon University Ph.D. Thesis, (2007).
- [6] A. Alexandrescu. *Modern C++ Design*. Addison-Wesley, 2001. Using concepts discussed in Chapter 2.
- [7] J. Järvi, J. Willcock and A. Lumsdaine. Following the `enable_if` family of templates contained in the `BOOST` libraries. <http://www.boost.org>.
- [8] W. Rarita and J. Schwinger. Phys. Rev. **60**, 61 (1941).
- [9] C. Zemach. Phys. Rev. **140**, B97 (1965).
- [10] S.U. Chung. BNL preprint BNL-QGS-02-0900 (2004).
- [11] A.V. Anisovich *et al.* J. Phys. G **28**, 15-32 (2002).
- [12] <http://www-meg.phys.cmu.edu/qft++>
- [13] C. Zemach. Phys. Rev. B **133**, 1201-1220 (1964).
- [14] O. Klein and Y. Nishina. Z. Physik **52**, 853 (1929).