Implementation of meson leptoproduction asymmetries in PARTONS

Kemal Tezgin

University of Connecticut

kemal.tezgin@uconn.edu

October 12, 2020

Writing cross-section in terms of asymmetries

```
PhysicalType<double> DVMPProcessGK06::CrossSection() {

    //check meson type
    if (m_mesonType != MesonType::PI0 && m_mesonType != MesonType::PIPLUS) {
        throw ElemUtils::CustomException(getClassName(), __func__,
                ElemUtils::Formatter() << "No implementation for meson "
                        << MesonType(m_mesonType).toString());
    }

    //result
    double result = Constant::FINE_STRUCTURE_CONSTANT
            * (m_W2 - pow(Constant::PROTON_MASS, 2))
            / (16 * pow(M_PI, 2) * pow(m_E, 2) * pow(Constant::PROTON_MASS, 2)
                    * m_Q2 * (1. - m_eps));

    result /= 32 * M_PI * (m_W2 - pow(Constant::PROTON_MASS, 2))
            * sqrt(lambdaFunction(m_W2, -m_Q2, pow(Constant::PROTON_MASS, 2)));

    result *= CrossSectionT() + m_eps * CrossSectionL();

    result *= 1. + AsymmetryAUUcosphi() * cos(m_phi) + AsymmetryAUUcos2phi() * cos(2*m_phi)
            + m_beamHelicity * AsymmetryALUsinphi() * sin(m_phi)
            + m_targetPolarization.getZ() * AsymmetryAULsinphi() * sin(m_phi)
            + m_targetPolarization.getZ() * AsymmetryAULsin2phi() * sin(2*m_phi)
            + m_targetPolarization.getZ() * AsymmetryAULsin3phi() * sin(3*m_phi)
            + m_beamHelicity * m_targetPolarization.getZ() * AsymmetryALLconst()
            + m_beamHelicity * m_targetPolarization.getZ() * AsymmetryALLcosphi() * cos(m_phi)
            + m_beamHelicity * m_targetPolarization.getZ() * AsymmetryALLcos2phi() * cos(2*m_phi);

    //apply dW2/dxB
    result *= m_Q2 / pow(m_xB, 2);

    //divide by 2pi to have dsigma/...dphiS
    result /= 2 * Constant::PI;

    return PhysicalType<double>(result, PhysicalUnit::GEVm2);
}
```

Asymmetries: $A_{UL}^{sin\phi}$

```
double DVMPProcessGK06::AsymmetryAULsinphi() {

    std::complex<double> amplitude0m0p = Amplitude0m0p();
    std::complex<double> amplitude0p0p = Amplitude0p0p();
    std::complex<double> amplitude0mpp = Amplitude0mpp();
    std::complex<double> amplitude0ppp = Amplitude0ppp();
    std::complex<double> amplitude0pmp = Amplitude0pmp();
    std::complex<double> amplitude0mmp = Amplitude0mmp();

    // sin(\phi) moment of A_{UL}. See Eq. (47) in arxiv:0906.0460

    double cosThetaGamma = sqrt(
            1.
                    - pow(m_gamma, 2) * (1. - m_y - pow(m_y * m_gamma / 2., 2))
                            / (1. + pow(m_gamma, 2)));
    double sinThetaGamma = sqrt(1. - pow(cosThetaGamma, 2));

    double sigma0 = 0.5 * (CrossSectionT() + m_eps * CrossSectionL());

    double A = std::imag(
            (std::conj(amplitude0ppp) + std::conj(amplitude0pmp))
                    * amplitude0p0p
                    + (std::conj(amplitude0mpp) + std::conj(amplitude0mmp))
                            * amplitude0m0p);

    double B = 2 * m_eps * std::imag(std::conj(amplitude0m0p) * amplitude0p0p);
    B -= m_eps * std::imag(std::conj(amplitude0mpp) * amplitude0ppp);
    B -= std::imag(
            std::conj(amplitude0mpp) * amplitude0pmp
                    - std::conj(amplitude0ppp) * amplitude0mmp);

    return (sinThetaGamma * B - sqrt(m_eps * (1. + m_eps)) * cosThetaGamma * A)
            / sigma0;
}
```

Implemented asymmetries

- $A_{UU}^{\cos(\phi)}$
- $A_{UU}^{\cos(2\phi)}$
- $A_{LU}^{\sin(\phi)}$
- $A_{UL}^{\sin(\phi)}$
- $A_{UL}^{\sin(2\phi)}$
- $A_{UL}^{\sin(3\phi)}$
- A_{LL}^{const}
- $A_{LL}^{\cos(\phi)}$
- $A_{LL}^{\cos(2\phi)}$

Asymmetries to be added next

$$
\begin{aligned}
A_{UT}^{\sin(\phi-\phi_s)}\sigma_0 &= -2\epsilon\cos\theta_\gamma\,\mathrm{Im}\Big[\mathcal{M}^*_{0-,0+}\mathcal{M}_{0+,0+}\Big] \\
&\quad -\cos\theta_\gamma\,\mathrm{Im}\Big[\mathcal{M}^*_{0+,++}\mathcal{M}_{0-,-+}-\mathcal{M}^*_{0-,++}\mathcal{M}_{0+,-+}\Big], \\
&\quad +\frac{1}{2}\sin\theta_\gamma\sqrt{\epsilon(1+\epsilon)}\,\mathrm{Im}\Big[(\mathcal{M}^*_{0+,++}+\mathcal{M}^*_{0+,-+})\mathcal{M}_{0+,0+} \\
&\quad +(\mathcal{M}^*_{0-,++}+\mathcal{M}^*_{0-,-+})\mathcal{M}_{0-,0+}\Big] \\
A_{UT}^{\sin(\phi_s)}\sigma_0 &= \cos\theta_\gamma\sqrt{\epsilon(1+\epsilon)}\,\mathrm{Im}\Big[\mathcal{M}^*_{0+,++}\mathcal{M}_{0-,0+}-\mathcal{M}^*_{0-,++}\mathcal{M}_{0+,0+}\Big], \\
A_{UT}^{\sin(2\phi-\phi_s)}\sigma_0 &= \cos\theta_\gamma\sqrt{\epsilon(1+\epsilon)}\,\mathrm{Im}\Big[(\mathcal{M}^*_{0+,-+}\mathcal{M}_{0-,0+}-\mathcal{M}^*_{0-,-+}\mathcal{M}_{0+,0+}\Big] \\
&\quad +\frac{1}{2}\epsilon\sin\theta_\gamma\,\mathrm{Im}\Big[\mathcal{M}^*_{0+,++}\mathcal{M}_{0+,-+}+\mathcal{M}^*_{0-,++}\mathcal{M}_{0-,-+}\Big], \\
A_{UT}^{\sin(\phi+\phi_s)}\sigma_0 &= \epsilon\cos\theta_\gamma\,\mathrm{Im}\Big[\mathcal{M}^*_{0-,++}\mathcal{M}_{0+,++}\Big] \\
&\quad +\frac{1}{2}\sin\theta_\gamma\sqrt{\epsilon(1+\epsilon)}\,\mathrm{Im}\Big[(\mathcal{M}^*_{0+,++}+\mathcal{M}^*_{0+,-+})\mathcal{M}_{0+,0+} \\
&\quad +(\mathcal{M}^*_{0-,++}+\mathcal{M}^*_{0-,-+})\mathcal{M}_{0-,0+}\Big] \\
A_{UT}^{\sin(2\phi+\phi_s)}\sigma_0 &= \frac{1}{2}\epsilon\sin\theta_\gamma\,\mathrm{Im}\Big[\mathcal{M}^*_{0+,++}\mathcal{M}_{0+,-+}+\mathcal{M}^*_{0-,++}\mathcal{M}_{0-,-+}\Big], \\
A_{UT}^{\sin(3\phi-\phi_s)}\sigma_0 &= \epsilon\cos\theta_\gamma\,\mathrm{Im}\Big[\mathcal{M}^*_{0+,-+}\mathcal{M}_{0-,-+}\Big]. \qquad (44)
\end{aligned}
$$

Outlook

- Users will be able to compute asymmetries for both π^0 and π^+ processes
- Cross section is written in terms of asymmetries