Technische Universität München
Fakultät für Physik

Abschlussarbeit im Bachelorstudiengang Physik

Parallelization of the computation of decay amplitudes for three-pion resonances with TensorFlow

Parallelisierung der Berechung von Zerfallsamplituden von 3-Pion-Resonanzen mit TensorFlow

3. September 2018

Erstgutachter (Themensteller): Prof. S. Paul
Zweitgutachter: Dr. O. Kortner

Contents

Introduction

Precision measurements of light-meson spectra help us to learn more about the strong interaction on energy scales where QCD perturbation theory is not applicable. Most hadrons are resonances, i.e. short-lived particles that decay via the strong interaction, often into multiple other particles which in turn decay as well. One method to analyse such multi-body decays is with the use of partial-wave analysis (PWA). By analysing the kinematic distribution of the daughter particles it is possible to reproduce mass, width and quantum numbers of the parent particles. Such Experiments produce huge amounts of data that, even with the help of computers, analysing these data in a manageable amount of time can be a very challenging task. ROOTPWA[1] is a software framework for partial-wave analysis and it was used to analyse data gathered by the COMPASS experiment at CERN. While ROOTPWA itself is highly optimized and can be used on single graphic cards, it is not able to run on a computer cluster, which becomes a necessity for certain analysis tasks. There are libraries which natively support the deployment of computations on computer clusters. One such libraries is the open-source framework TensorFlow[2], which was originally developed by Google. In this thesis, the possibility of incorporating TensorFlow into ROOTPWA is investigated. The main focus lies on the implementation of a key feature of ROOTPWA, namely the computation of the so-called decay amplitudes, using TensorFlow. These amplitudes describe the decay of states with well-defined quantum numbers into multi-body final states. We will verify computed results for a specific partial wave amplitude and investigate how the computation scales with the number of CPU cores on a single workstation.

[1] `https://github.com/ROOTPWA-Maintainers/ROOTPWA`
[2] `https://www.TensorFlow.org`

Chapter 1

Partial-Wave Formalism

The partial-wave formalism (also known as partial-wave decomposition or partial-wave analysis) is a mathematical method to analyse multi-body decays of hadrons that were produced, for example, in scattering experiments. In this approach the measured kinematic distribution of the final state particles is decomposed into amplitudes with well-defined angular-momentum quantum number. The theoretical approach used by ROOTPWA is explained in the following to the point where decay amplitudes are introduced. After that we will focus on these amplitudes in order to derive a formula to compute them for a chosen partial wave. The deriviation presented here follows ref. [1].

1.1 Ansatz

In this work, we consider inelastic scattering processes of the form

$$a + b \to (1 + 2 + \cdots + n) + c, \tag{1.1}$$

where a is a high-energetic beam hadron, b is the target hadron which strongly interacts with a thereby producing n hadrons $(1, 2, \ldots, n)$ in the final state and a target recoil c. Such reactions are studied at the COMPASS experiment at CERN, where a high energy π^- beam is shot at a proton target producing mostly π, K, η and η' in the final state. These reactions are well suited to study the excitation spectrum of light-quark mesons. In this work, we will only be concerned with one reaction where three charged pions are produced in the final state. But let us first look at most the general case.

The cross section for processes like eq. (1.1) is given by

$$\mathrm{d}\sigma_{a+b\to(1+2+\cdots+n)+c} = \frac{1}{4\sqrt{(p_a \cdot p_b)^2 - m_a^2 m_b^2}} \left|\mathcal{M}_{fi}\right|^2 \mathrm{d}\Phi_{n+1}\left(p_a + p_b; p_1, \ldots, p_n, p_c\right), \tag{1.2}$$

where the p_i are the four-momenta and the m_i are the masses of all involved particles [2]. The Lorentz-invariant transition matrix element from the initial to

the final state is represented by $\mathcal{M}_{fi}$. We assume that the term $|\mathcal{M}_{fi}|^2$ contains the incoherent averaging and summation over the spin states of the initial and final state. The Lorentz-invariant $(n+1)$-body phase-space element of the final-state particles is represented by $\mathrm{d}\Phi_{n+1}$. One way to model reactions like in eq. (1.1) is to assume that the n-body final state is produced via a t-channel exchange process and that intermediate n-body states are dominated by resonances. This means we can rewrite eq. (1.1) into a two-stage process, with the first process being the production of a resonance X in a two-body inelastic scattering reaction

$$a + b \to X + c \tag{1.3}$$

and the second process being the subsequent decay of X into an n-body hadronic final state

$$X \to 1 + 2 + \cdots + n. \tag{1.4}$$

The whole reaction is shown in fig. 1.1.

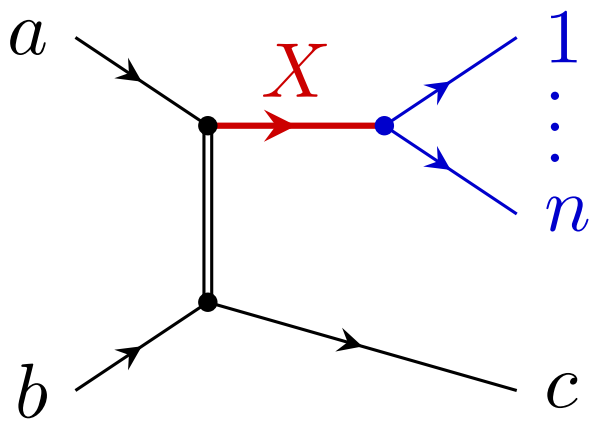


Figure 1.1: Scattering reaction as described by eq. (1.1)

The inelastic two-body scattering process in eq. (1.3) is described by the n-body invariant mass m_X and the two Mandelstam variables s and t, where s is the squared center-of-mass energy of the (a, b) system and t is the transferred squared four-momentum. Using these variables we can rewrite eq. (1.2) into

$$\begin{aligned}&\mathrm{d}\sigma_{a+b\to(1+2+\cdots+n)+c}\\ &= \frac{1}{16\pi}\frac{1}{\left[s-(m_a+m_b)^2\right]\left[s-(m_a-m_b)^2\right]}\,|\mathcal{M}_{fi}|^2\,\mathrm{d}t'\,\frac{2m_x}{2\pi}\,\mathrm{d}m_X\,\mathrm{d}\Phi_n\left(p_X; p_1, \ldots, p_n\right).\end{aligned} \tag{1.5}$$

Here, we have split the phase-space element $\mathrm{d}\Phi_{n+1}$ into a two-body phase-space element of X and c and a n-body phase-space element $\mathrm{d}\Phi_n$ for the decay of X. Since a decay conserves the total momentum we can write $\mathrm{d}\Phi_n$ as

$$\mathrm{d}\Phi_n\left(p_X; p_1, \ldots, p_n\right) = \delta^{(4)}\left(p_X - \sum_{i=1}^{n} p_i\right) \prod_{i=1}^{n} \frac{\mathrm{d}^3 p_i}{(2\pi)^3 2E_i}. \tag{1.6}$$

Also note that instead of t we use in eq. (1.5) the reduced squared four-momentum transfer t' which is defined as

$$t' = |t| - |t|_{\min} \geq 0. \tag{1.7}$$

Here $|t|_{\min}$ is the minimum absolute value of t required to produce the (X, c) system. From here on, the s dependency will be dropped from the equations because the beam energy of the studied reaction was fixed. Hence, the kinematics of the reaction is completely defined by m_X, t' and an additional set of $(3n-4)$ phase-space variables[1]. The phase-space variables fully describe the n-body system and are represented by τ_n.

Instead of the cross section σ the experiment measures intensity distribution $\mathcal{I}$. A detector with unity acceptance over the full kinematic range would measure a intensity distribution

$$\begin{aligned}\mathcal{I}\left(m_X, t', \tau_n\right) = \frac{\mathrm{d}N}{\mathrm{d}\Phi_{n+1}} = \frac{\mathrm{d}N}{\mathrm{d}m_X\, \mathrm{d}t'\, \mathrm{d}\Phi_n\left(m_X, \tau_n\right)} &\propto \frac{\mathrm{d}\sigma_{a+b\to(1+2+\cdots+n)+c}}{\mathrm{d}m_X\, \mathrm{d}t'\, \mathrm{d}\Phi_n\left(m_X, \tau_n\right)} \\ &\propto m_X \left|\mathcal{M}_{fi}\left(m_X, t', \tau_n\right)\right|^2, \end{aligned} \tag{1.8}$$

where N is the number of detected events and $\mathrm{d}\Phi_n\left(m_X, \tau_n\right)$ is the differential n-body phase-space element from eq. (1.6) with the Dirac delta function integrated out. It is worth pointing out that we dropped all constants from eq. (1.5) in eq. (1.8) and since $\mathcal{I}$ is differential in the Lorentz-invariant phase-space element $\mathrm{d}\Phi$, it is independent of the choice of the phase-space variables τ_n. Basically, $\mathcal{I}$ is a measure for the deviation of the measured kinematic distribution from a pure phase-space distribution and is therefore a direct measure for $\left|\mathcal{M}_{fi}\left(m_X, t', \tau_n\right)\right|^2$. From here on, matrix elements will be referred to as amplitudes.

Assuming that the intermediate states X are dominated by resonances, we can factorize the amplitude for X into two parts: (i) the so-called transition amplitude $\mathcal{T}_i(m_X, t')$ that describes the production and propagation of the state X and (ii) the decay amplitude $\Psi_i(m_X, \tau_n)$ that describes the decay of X into a particular n-body

[1] The $4n$ components of the four-momenta of the n final-state particles, which are constrained by their n known masses and another 4 equations, which account for energy conservation and the known four-momentum of X, leave $4n - n - 4 = 3n - 4$ unknown variables.

finals state. The index i summarizes the set of quantum numbers that uniquely define the process in eq. (1.3) and eq. (1.4). It will be discussed further in 1.2.4. In the scattering reactions considered here, the possible quantum numbers for X are limited only by the conservation laws of the strong interaction. In addition, the X may decay via various decay chains. Hence usually many amplitudes with various i contribute to the intensity. Since the initial and final state particles for all these amplitudes are the same, the amplitudes may interfere with each other and hence have to be added coherently. Therefore, the model for the intensity distribution can be written as

$$\mathcal{I}(m_X, t', \tau_n) = \left| \sum_{i}^{\text{waves}} \mathcal{T}_i \left(m_X, t'\right) \Psi_i \left(m_X, \tau_n\right) \right|^2 . \tag{1.9}$$

Using model assumptions, the decay amplitudes can be calculated. However, the transition amplitudes are unknown. It is actually the goal of the PWA to extract them from the data. To this end, the data are subdivided into narrow bins in m_X and t'. In each (m_X, t') cell, the PWA model in eq. (1.9) is fit to the measured five-dimensional intensity distribution using an extended maximum likelihood approach. The free parameters in this fit are the set of transition amplitudes $\{\mathcal{T}_i\}$ in the given (m_X, t') bins. In a second stage of the analysis, one can then search for resonance signals in the selected transition amplitude.

1.2 Decay Amplitude

To calculate the decay amplitudes $\Psi_i \left(m_X, \tau_n\right)$, we utilize the so-called isobar model [3, 4]. In this model, the decay of X is described as a chain of successive two-body decays with the appearance of additional intermediate resonances. Another fundamental assumption is that the outgoing particles of the successive decays do not interact with each other, i.e. we do not consider final-state interactions.

To construct a formula for $\Psi_i \left(m_X, \tau_n\right)$, we have to first construct a formula for the two-body decay amplitude $\mathcal{A}_r^{J_r M_r}$. This amplitude describes the propagation of a resonance r with spin J_r and spin projection M_r onto a chosen quantization axis and the decay of r into particle 1 and 2. The two body-body decay amplitude can be calculated in the r rest frame using the helictiy formalism [5]. The daughter particle have spins $J_{1,2}$ and are described in helicity bases where the quantization axes are along the directions of the momenta of the respective particles. Due to conservation of momentum the momenta of the particles will be back to back in the r rest frame and have a magnitude given by the mass of r and the daughter masses. This allows to completely define the kinematics of the decay by the polar angle θ and the azimuthal angle ϕ of one daughter.

The daughter particles will be described as a two-particle plane-wave center-of-mass helicity state $|\boldsymbol{p_1}, \boldsymbol{p_2}; \lambda_1, \lambda_2\rangle$ where $\lambda_{1,2}$ is the helicity of the respective daughter particle and $\boldsymbol{p_1} = -\boldsymbol{p_2} = \boldsymbol{q}$ its momentum, where $\boldsymbol{q}$ is the breakup momentum of r. The magnitude of the breakup momentum $\boldsymbol{q}$ is given by

$$\begin{aligned} |\boldsymbol{q}|^2 &= q^2\left(m_r, m_1, m_2\right) \\ &= \frac{\left[m_r^2 - \left(m_1^2 + m_2^2\right)\right]\left[m_r^2\left(m_1^2 - m_2^2\right)\right]}{4m_r^2} \end{aligned} \tag{1.10}$$

The breakup momentum only depends on the particle masses and is hence a constant. This allows us to write the previous quantum states as $|\theta_r, \phi_r; \lambda_1, \lambda_2\rangle$. We can finally write the two-body decay amplitude of a resonance r with mass m_r and spin state $|J_r, M_r\rangle$ as

$$\mathcal{A}_r^{J_r M_r}\left(m_r, \theta_r, \phi_r\right) = \mathcal{D}_r\left(m_r\right) \sum_{\lambda_1, \lambda_2} \langle \theta_r, \phi_r; \lambda_1, \lambda_2 | \hat{T}(m_r) | J_r, M_r \rangle \tag{1.11}$$

where $\mathcal{D}_r(m_r)$ represents the propagation of r and $\hat{T}(m_r)$ the transition operator of the decay. The coherent sum over all allowed daughter helicities will only be executed if both daughter particles are intermediate states in the decay chain. If at least on of the daughters is a (quasi-stable) final state particle and their helicity was not measured, then the summation over the respective helicities has to be performed incoherently on the cross-section level.

We now expand the two-body decay amplitude into partial waves by inserting a complete set of angular-momentum helicity states $|J, M, \lambda_1, \lambda_2\rangle$, which describe a two-particle state with definite spin. The amplitude now reads

$$\begin{aligned} \mathcal{A}_r^{J_r M_r}\left(m_r, \theta_r, \phi_r\right) = \mathcal{D}_r\left(m_r\right) \sum_{\lambda_1, \lambda_2} & \langle \theta, \phi; \lambda_1, \lambda_2 | J_r, M_r, \lambda_1, \lambda_2 \rangle \\ & \times \langle J_r, M_r, \lambda_1, \lambda_2 | \hat{T}(m_r) | J_r, M_r \rangle . \end{aligned} \tag{1.12}$$

The amplitude can be further expanded into two-particle states $|J_r, M_r, L_r, S_r\rangle$, which describe states that have definite relative orbital angular momentum L_r between the two daughters and where the spins of both daughters couple to the total intrinsic spin S_r:

$$\begin{aligned} \mathcal{A}_r^{J_r M_r}\left(m_r, \theta_r, \phi_r\right) = \mathcal{D}_r\left(m_r\right) \sum_{\lambda_1, \lambda_2} & \overbrace{\langle \theta_r, \phi_r; \lambda_1, \lambda_2 | J_r, M_r, \lambda_1, \lambda_2 \rangle \langle J_r, M_r, \lambda_1, \lambda_2 | J_r, M_r, L_r, S_r \rangle}^{\text{angular part}} \\ & \times \underbrace{\mathcal{D}_r\left(m_r\right) \langle J_r, M_r, L_r, S_r | \hat{T}(m_r) | J_r, M_r \rangle}_{\text{dynamical part}} . \end{aligned} \tag{1.13}$$

We see that the amplitude factorizes into an angular part which is given by first principles and completely defined by the angular-momentum quantum numbers and the decay angles and a dynamical part, which only depends on the invariant mass m_r of the $(1,2)$ system and which we need to model.

1.2.1 Angular Part

The first scalar product in the angular part in eq. (1.13) describes the angular distribution of the daughter particles in the r rest frame frame. It can be written as [5, 6]

$$\langle J_r, M_r, \lambda_1, \lambda_2 | \theta_r, \phi_r; \lambda_1, \lambda_2 \rangle = \sqrt{\frac{2J_r+1}{4\pi}} D^{J_r}_{M_r \lambda} (\phi_r, \theta_r, 0) \,, \tag{1.14}$$

where D represents the Wigner D-function [7]. Note that λ will be defined further below. This function describes the transformation property of a spin state $|J, M\rangle$ under an active rotation $\hat{\mathcal{R}}$ defined by the three Euler angles α, β and γ. Since the $|J, M\rangle$ basis is complete, any rotated state can be expressed as a linear combination of these basis states like

$$\begin{aligned} \hat{\mathcal{R}}(\alpha,\beta,\gamma) |J, M\rangle &= \sum_{M'=-J}^{+J} \left|J, M'\right\rangle \overbrace{\left\langle J, M' \right| \hat{\mathcal{R}}(\alpha,\beta,\gamma) |J, M\rangle}^{D^J_{M'M}(\alpha,\beta,\gamma)} \\ &= \sum_{M'=-J}^{+J} D^J_{M'\,M}(\alpha,\beta,\gamma) \left|J, M'\right\rangle \end{aligned} \tag{1.15}$$

Using the same convention as in ref. [8], where

$$\hat{\mathcal{R}}(\alpha,\beta,\gamma) = e^{-i\alpha \hat{J}_z} e^{-i\beta \hat{J}_y} e^{-i\gamma \hat{J}_z} \tag{1.16}$$

and $\hat{J}_i$ is the ith component of the angular momentum operator, yields

$$\begin{aligned} D^J_{M'\,M}(\alpha,\beta,\gamma) &= \left\langle J, M' \right| e^{-i\alpha \hat{J}_z} e^{-i\beta \hat{J}_y} e^{-i\gamma \hat{J}_z} |J, M\rangle \\ &= e^{-iM'\alpha} \left\langle J, M' \right| e^{-i\beta \hat{J}_y} |J, M\rangle e^{-iM\gamma} \\ &= e^{-iM'\alpha} d^J_{M'\,M}(\beta) e^{-iM\gamma} \end{aligned} \tag{1.17}$$

with the small Wigner d-function

$$\begin{aligned} d^J_{M'\,M}(\beta) = &\left[(J+M)!\,(J-M)!\,\left(J+M'\right)!\,\left(J-M'\right)!\right]^{\frac{1}{2}} \\ &\times \sum_k \frac{(-1)^k}{k!\,(J+M-k)!\,(J-M'-k)!\,(M'-M+k)!} \\ &\times \left(\cos\frac{\beta}{2}\right)^{2J+M-M'-2k} \left(\sin\frac{\beta}{2}\right)^{M'-M+2k} . \end{aligned} \tag{1.18}$$

The sum has to performed for those values of k for which the factorials are non-negative.

Now we are able to actually compute the Wigner D-function and we can derive eq. (1.14). To this end, we consider the special case where $\theta_r = \phi_r = 0$, i.e. the daughter particles move in opposite directions along the quantization axis. In this case, the two-particle plane-wave center-of-mass helicity state $|0,0;\lambda_1,\lambda_2\rangle$ is an eigenstate of $\hat{J}_z$. The total spin projection onto the z is $\lambda = \lambda_2 - \lambda_1$. We can assume, without loss of generality, that particle 1 moves in the $+z$ direction. The two-particle state can be expressed in terms of angular-momentum helicity states with

$$|0,0;\lambda_1,\lambda_2\rangle = \sum_{J_r=0}^{\infty} |J_r,\lambda;\lambda_1,\lambda_2\rangle \underbrace{\langle J_r,\lambda;\lambda_1,\lambda_2|0,0;\lambda_1,\lambda_2\rangle}_{C_{J_r}} . \tag{1.19}$$

Based on this state we can construct an arbitrary two-particle plane-wave center-of-mass helicity state by applying an active rotation $\hat{\mathcal{R}}(\phi_r,\theta_r,0)$, where ϕ_r and θ_r describe the direction of particle 1:

$$|\phi_r,\theta_r;\lambda_1,\lambda_2\rangle = \hat{\mathcal{R}}(\phi_r,\theta_r,0)\,|0,0;\lambda_1,\lambda_2\rangle = \sum_{J_r=0}^{\infty} C_{J_r}\hat{\mathcal{R}}(\phi_r,\theta_r,0)\,|J_r,\lambda;\lambda_1,\lambda_2\rangle\,. \tag{1.20}$$

The coefficients C_{J_r} are fixed by the two-particle states and the normalization of the Wigner D-functions. With eq. (1.15) we can now write

$$|\theta_r,\phi_r;\lambda_1,\lambda_2\rangle = \sum_{J_r=0}^{\infty}\sum_{M_r=-J_r}^{+J_r} \sqrt{\frac{2J_r+1}{4\pi}} D^{J_r}_{M_r,\,\lambda}(\phi_r,\theta_r,0)\,|J_r,M_r;\lambda_1,\lambda_2\rangle\,. \tag{1.21}$$

Using this equation one can compute the scalar product in eq. (1.14) and verify the identity.

The second scalar product in the angular part of eq. (1.13) is called the recoupling coefficient. It connects the two-particle angular-momentum states in the LS-coupling and helicity representation and is given by [5]

$$\langle J_r,M_r;L_r,S_r|J_r,M_r;\lambda_1,\lambda_2\rangle = \sqrt{\frac{2L_r+1}{2J_r+1}}\underbrace{(J_1,\lambda_1;J_2,-\lambda_2|S_r,\lambda)}_{\text{spin-spin coupling}}\underbrace{(L_r,0;S_r,\lambda|J_r,\lambda_r)}_{\text{spin-orbit coupling}} \tag{1.22}$$

The two appearing factors are Clebsch-Gordan coefficients. The first one for the coupling of the spins of the daughter particles to the total intrinsic spin S_r and the second one for the coupling of L_r and S_r to J_r. It should be noted that by construction the orbital angular momentum L_r is perpendicular to the momenta of both daughters and has therefore no projection onto the helicity quantization axis.

1.2.2 Dynamical Part

First, we look at the propagator term $\mathcal{D}_r(m_r)$ of the resonance r. For many resonances, as well as for the one we will discuss later on, the propagator can be parametrized using a relativistic Breit-Wigner amplitude of the form

$$\mathcal{D}_r\left(m_r; m_0, \Gamma_0\right) = \frac{m_0\Gamma_0}{m_0^2 - m_r^2 - im_0\Gamma(m_r)}, \tag{1.23}$$

where m_0 and Γ_0 are the nominal mass and width of the resonance. In the simplest case, i.e. for a narrow resonance, we can assume the total width to be constant, i.e.

$$\Gamma(m_r) = \Gamma_0 \tag{1.24}$$

For a wide resonance we need a better approximation:

$$\Gamma(m_r) = \sum_{i}^{\text{decay modes}} \Gamma_i \frac{q_i}{m_r} \frac{m_0}{q_{i,0}} \frac{F_{L_i}^2(q_i)}{F_{L_i}^2(q_{i,0})} \tag{1.25}$$

which sums over all decay modes i of resonance r. The total width depends on the partial width Γ_i of each decay channel i, the magnitude of the breakup momentum $q_i = q(m_r; m_{i,1}, m_{i,2})$ as given by eq. (1.10), the breakup momentum of the resonance mass $q_{i,0} = q(m_0; m_{i,1}, m_{i,2})$ and the centrifugal-barrier factor $F_{L_i}(q_i)$, which parametrizes the orbital angular-momentum barrier.

The centrifugal-barrier factor $F_L(q)$ takes into account the barrier effect that is caused by the orbital angular momentum between the daughters in a two-body decay. The maximal value of L is limited by the magnitude of the breakup momentum q and the impact parameter d between the daughter particles. The latter is generally assumed to be given by the range of the strong interaction, which is about 1 fm. This corresponds to a range parameter of $q_r = 197$ MeV/c. A phenomenological parametrization of the barrier factors was derived by Blatt and Weiskopf in ref. [9] by solving the non-relativistic Schrödinger equation for a square-well potential with a radius given by the range of the strong interaction, which corresponds to the impact parameter. The solutions are the spherical Hankel functions of the first kind. The barrier factors are usually written as functions of a dimensionless variable

$$z(m_r) = \left[\frac{q(m_r)}{q_r}\right]^2. \tag{1.26}$$

Using the parametrization from ref. [10] yields

$$F_L^2(z) = \frac{1}{z\left|h_L^{(1)}(\sqrt{z})\right|^2}, \tag{1.27}$$

with the spherical Hankel function of the first kind given by

$$h_L^{(1)} = (-i)^{L+1} \frac{e^{ix}}{x} \sum_{k=0}^{L} \frac{(L+k)!}{(L-k)!k!} \left(\frac{i}{2x}\right)^k. \tag{1.28}$$

It is customary to normalize the barrier factors such that $F_L(1) = 1$. The barrier factors for the lowest values of L up to 3 are[2]

$$F_0^2(z) = 1, \tag{1.29}$$

$$F_1^2(z) = \frac{2z}{z+1}, \tag{1.30}$$

$$F_2^2(z) = \frac{13z^2}{z^2+3z+1}, \tag{1.31}$$

$$F_3^2(z) = \frac{277z^3}{z^3+6z^2+45z+225}. \tag{1.32}$$

Decays with large values of L are usually suppressed so that higher orders of $F_L(z)$ are rarely needed.

Lastly, we need to parametrize the second factor of the dynamical part in eq. (1.13), i.e. the expectation value of the transition operator. It is given by

$$\langle \theta_r, \phi_r; \lambda_1, \lambda_2 | \hat{T}(m_r) | J_r, M_r \rangle = \alpha_{r\to 1+2} F_{L_r}(m_r), \tag{1.33}$$

where $\alpha_{r\to 1+2}$ is a complex-valued coupling constant which describes strength and relative phase of the decay mode. This parametrization is a further assumption needed to make the decay amplitudes computable. This coupling constant will be treated as an additional fit parameter and it will be, like all fit parameters, absorbed by $\mathcal{T}_{a+b\to X+c}(m_X, t')$ from eq. (1.9). Therefore, $\alpha_{r\to 1+2}$ will no longer be part of the following equations.

1.2.3 Formula for n-body decay

Now we have all ingredients for the two-decay amplitude in eq. (1.13). This means we can write down a recursive formula to calculate the decay amplitude for an n-

[2] It is sensible to write these functions out, because implementing the Hankel functions is not only tedious, they are computationally expensive.

-body decay. Inserting eqs. (1.14), (1.22) and (1.33) into eq. (1.13) yields

$$\begin{aligned}\mathcal{A}_r^{J_r M_r L_r S_r}(m_r,\theta_r,\phi_r) = \underbrace{\sqrt{\frac{2L_r+1}{4\pi}}}_{\text{normalization}} \overbrace{\underbrace{\mathcal{D}_r(m_r)}_{\text{propagator}}\ \underbrace{F_{L_r}(m_r)}_{\text{barrier factor}}}^{\text{dynamical part}} \\ \times \sum_{\lambda_1,\lambda_2} \underbrace{(J_1,\lambda_1;J_2,-\lambda_2|S_r,\lambda)\,(L_r,0;S_r,\lambda|J_r,\lambda_r)\,D^{J_r\,*}_{M_r\lambda_r}(\phi_r,\theta_r)}_{\text{angular part}} \\ \times \underbrace{\mathcal{A}_1^{J_1 M_1 L_1 S_1}(m_1,\theta_1,\phi_1)}_{\text{decay of 1}} \underbrace{\mathcal{A}_2^{J_2 M_2 L_2 S_2}(m_2,\theta_2,\phi_2)}_{\text{decay of 2}},\end{aligned} \tag{1.34}$$

where we, compared to eq. (1.13), added the amplitudes $\mathcal{A}_1^{J_1 M_1 L_1 S_1}$ and $\mathcal{A}_2^{J_2 M_2 L_2 S_2}$. These terms account for the further decays of the daughter particles. If the respective daughter particle is stable, decay amplitude becomes unity. In this case, the amplitude of the daughter particle has the same form as eq. (1.34). In essence, to compute the n--decay amplitude eq. (1.34) has to be applied recursively for each two--body vertex in the decay chain.

1.2.4 Concrete Example

Using eq. (1.34) we can derive an explicit formula for the decay amplitude for a specific partial-wave which occurs in the reaction

$$p^+ + \pi^- \to p^+ + \pi^+ + \pi^- + \pi^-. \tag{1.35}$$

Figure 1.2 shows the process by using a specific partial wave as an example.

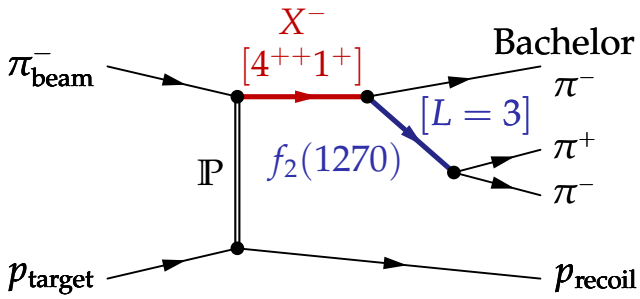


Figure 1.2: Scattering reaction in case of the $4^{++}1^+\, f_2\,(1270)\ \pi\, F$ partial wave.

Applying eq. (1.34) recursively and using that pions are spinless yields[3]

$$\Psi_i(m_X, \overbrace{\theta_{\mathrm{GJ}}, \phi_{\mathrm{GJ}}, m_r, \theta_{\mathrm{HF}}, \phi_{\mathrm{HF}}}^{\tau_3}) = \sqrt{\frac{2L_X+1}{4\pi}} F_{L_X}(m_X)$$
$$\times \sum_{\lambda_r} (J_r, \lambda_r; 0, 0 | S_X, \lambda_X)(L_X, 0; S_X, \lambda_X | J, \lambda_X) D^{J\,*}_{M\lambda_r}(\theta_{\mathrm{GJ}}, \phi_{\mathrm{GJ}}, 0)$$
$$\times \sqrt{\frac{2J_r+1}{4\pi}} F_{J_r}(m_r)\mathcal{D}_r(m_r) D^{J_r\,*}_{\lambda_r 0}(\theta_{\mathrm{HF}}, \phi_{\mathrm{HF}}, 0). \tag{1.36}$$

The angles carry indices GJ and HF, which denote the reference frame in which they are defined. Both reference frames are right-handed coordinate systems and they are the rest frames of the respective parent particle whose decay we want to describe. GJ stands for the Gottfried-Jackson frame and is the rest frame of X. In this reference frame the particle beam defines the z_{GJ} axis and the y_{GJ} axis is given by the normal of the production plane: $\hat{\boldsymbol{y}}_{\mathrm{GJ}} \propto \hat{\boldsymbol{p}}_a^{\mathrm{lab}} \times \hat{\boldsymbol{p}}_X^{\mathrm{lab}} \propto \hat{\boldsymbol{p}}_a^{\mathrm{GJ}} \times \hat{\boldsymbol{p}}_a^{\mathrm{GJ}}$. HF stands for the helicity reference frame where the $\pi^+\pi^-$ isobar resonance is at rest. The coordinate system is defined by taking the z_{HF} along the original direction of the motion of the isobar and $\hat{\boldsymbol{y}}_{\mathrm{HF}} \propto \hat{\boldsymbol{z}}_{\mathrm{GJ}} \times \hat{\boldsymbol{z}}_{\mathrm{HF}}$. Normally, the angles would have to extracted form the raw data by performing several Lorentz-transformations. Since we are only concerned about the computation of the decay amplitude, the angles will be provided in the data set.

If we compare eq. (1.36) with fig. 1.2 we see that even though there are two decay vertices, there is only one sum over the helicity states of the isobar r and only two Clebsch-Gordan coefficients. Fortunately, pions are spinless particles which causes one sum and two Clebsch-Gordan coefficients to be reduced to unity. We also see that there is no propagator $\mathcal{D}_X$. Looking back to eq. (1.9), this propagator was defined to be part of transition amplitudes $\mathcal{T}_i$ is therefore omitted from this equation. Equation (1.36) can be further simplified to

$$\Psi_i(m_X, \theta_{\mathrm{GJ}}, \phi_{\mathrm{GJ}}, m_r, \theta_{\mathrm{HF}}, \phi_{\mathrm{HF}}) =$$
$$\underbrace{\frac{\sqrt{(2L_X+1)(2J_r+1)}}{4\pi} F_{L_X}(m_X) F_{J_r}(m_r)\mathcal{D}_r(m_r)}_{\text{dynamical part}}$$
$$\times \underbrace{\sum_{\lambda_r} (L_X, 0; J_r, \lambda_r | J, \lambda_r) D^{J\,*}_{M\lambda_r}(\theta_{\mathrm{GJ}}, \phi_{\mathrm{GJ}}, 0) D^{J_r\,*}_{\lambda_r 0}(\theta_{\mathrm{HF}}, \phi_{\mathrm{HF}}, 0)}_{\text{angular part}}. \tag{1.37}$$

[3] The index i holds by definition all relevant quantum numbers of a particular partial-wave. The short-hand notation of the given example is $i = 4^{++}1^+ f_2(1270)\,\pi\,F$. Note that F is the spectroscopic notation for 3.

Since pions are bosonic particles and there are two negatively charged indistinguishable pions in the final state we would normally have to symmetrize the decay amplitude under exchange of the two π^-. Since here we are only concerned with proof-of-principal computation of the decay amplitude, we will leave eq. (1.37) as is and will implement using TensorFlow.

The partial-wave we will use to evaluate the performance of the Tensorflow implementation is the partial-wave shown in fig. 1.2. The involved particles as well as their quantum numbers are:

$$\begin{aligned} X: \quad & J_X^{P_X C_X} M_X^{\epsilon_X} = 4^{++}1^{+} \\ f_2(1270): \quad & J_r^{P_r C_r} = 2^{++} \\ & L_X = 3, \end{aligned}$$

X is the resonance produced in the scattering reaction, $f_2\,(1270)$ is an intermediate isobar $\pi^+\pi^-$ resonance in the decay chain and L_X is the orbital angular-momentum between the bachelor pion and $f_2\,(1270)$. Note that the spin projection of X is given with two quantum numbers. M_X is the absolute value of the projection and ϵ_X, the so-called reflectivity, is the sign of the projection.

Chapter 2

TensorFlow

TensorFlow is an open-source software library for high-performance numerical computations. It was especially designed for machine-learning applications and the implementation of artificial neural networks (ANN). This design choice heavily impacts the way, in which algorithms can be implemented with TensorFlow. Therefore, we will briefly discuss the fundamentals of ANNs [11] to understand how to implement numerical computations with TensorFlow. Then we look at the explicit implementation of eq. (1.37).

2.1 Artificial Neural Network

The building blocks of ANNs are so-called perceptrons as shown in fig. 2.1. A

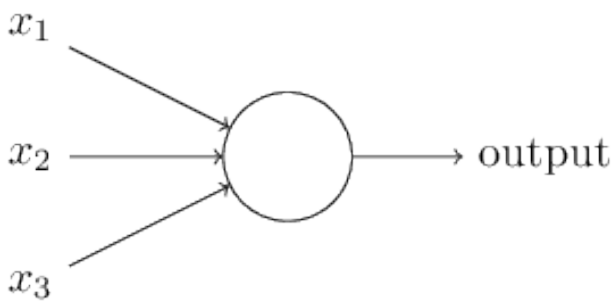


Figure 2.1: Schematic diagram of a perceptron [11].

perceptron is a symbolic representation of a mathematical operation that can taken an arbitrary amount of input variables x_i and produces one mathematical value as an output. The most common mathematical operation represented by a perceptron in ANNs is

$$\text{output} = \begin{cases} 0 & \text{if} \quad \sum_i w_i \cdot x_i - b \leq 0, \\ 1 & \text{otherwise,} \end{cases} \quad , \tag{2.1}$$

where x_i are the input values, w_i are the so-called weights and b is the so-called bias. Each perceptron carries their own weights and biases. ANNs are created by interconnecting perceptrons with one another. Such networks can consist of just a few or up to several thousands perceptrons. This means that the mathematical logic of an algorithm is projected onto a graph of interconnected nodes where each one represents a mathematical operation. This philosophy is fully embraced by TensorFlow. Algorithms are implemented as computational TensorFlow graphs. This is the reason why eq. (1.37) was explicitly derived instead of simply implementing and using eq. (1.34). A recursive algorithm can not be implemented with TensorFlow, since the computational graph has to be explicitly defined before the start the computation. At first glance this may seem like a disadvantage, but since the graph is explicitly defined it can be modified, i.e optimized, before the actual computation. ROOTPWA lacks such a feature. One way to optimize a graph is by removing redundant operations. A redundant operation could be a node whose input never changes. So instead of computing the node every time the graphs is called, the node can be computed once and than be replaced by a node which just carries a constant value equal to the just computed one. Looking back at eq. (1.37), all terms which only depend on the quantum numbers and the nominal masses, like the Clebsch-Gordan coefficients, are in terms the computation, redundant. It should be noted that graph modification is still an experimental feature in TensorFlow, and though I have successfully tested preprocessing of computation graphs, this feature will not be used in the performance studies in section 3.2.

Lastly, to actually use an ANN, it needs to be trained. To train a network training data, consisting of input and corresponding output values, has to given. Based on the training data, the weights and biases are changed in order to minimize the difference between given and computed output values. This is an iterative process which is actually very similar to fitting data. The main focus of this thesis is the the computation of decay amplitudes, but the main purpose of ROOTPWA is to extract information by fitting a theoretical model onto the data set. Since training a neural network, i.e. data fitting, is the most essential part of machine-learning, TensorFlow offers many features for this process. These features can be used to possibly improved the performance of ROOTPWA even further. This is one of the reasons why TensorFlow was chosen to be incorporated with ROOTPWA.

2.2 Implementation of the decay amplitude with TensorFlow

The nodes of a TensorFlow graph can be bundled together into so called subgraphs. Subgraphs themselves can also be divided into further subgraphs. Using this concept

allows to construct layered TensorFlow graphs. The structure of the graph can then be easily visualized by a TensorFlow feature called TensorBoard. It does not only visualize the graph itself but even how the data figuratively flows through the graph. We will now briefly examine how eq. (1.37) is implemented as a computation graph. The top-most layer of the graph is shown in fig. 2.2. The graph has been structured

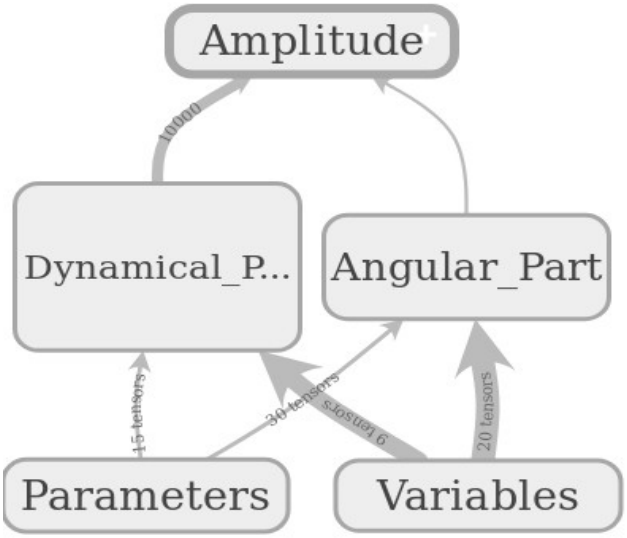


Figure 2.2: Top-most layer of the TensorFlow graph representation of formula eq. (1.37)

in such a way that the dynamical and angular part are computed by a their own subgraphs. The expanded subgraphs of both parts are shown in figs. A.1 and A.2 The input nodes have also been separated. The subgraph Parameters contains all nodes that hold constant values, i.e. quantum numbers, nominal masses and widths and the subgraph Variables holds all the nodes that can be fed data to, i.e. the kinematic variables $\tau_3 = \{m_X, \theta_{\mathrm{GJ}}, \phi_{\mathrm{GJ}}, m_r, \theta_{\mathrm{HF}}, \phi_{\mathrm{HF}}\}$. The arrows between the subgraphs indicate how many tensors, i.e. how much data, flows from one subgraph to another. For example, the arrow connecting the subgraph Variables with subgraph Angular_Part shows that 20 tensors flow from Variables to Angular_Part. Looking back at eq. (1.37), we see Angular_Part represents the sum over two Wiger D-functions which depend on two angles, i.e. two nodes in the subgraph Variables, each. The sum goes from $\lambda_r = -2$ to $\lambda_r = +2$, so we need $2 \cdot 2 \cdot 5 = 20$ tensors from Variables to compute Angular_Part, hence the flow of 20 tensors.

Tensors in this context are multi-dimensional arrays which can have an arbitrary shape. Since TensorFlow allows vectorization, i.e. we can group our data into

arrays and give the whole array to the TensorFlow graph and it will project the mathematical operation onto all data points in the array accordingly, it is not necessary to manually loop over every single data point and feed it to the graph directly. Because there a six kinematic variables, i.e. $\{m_X, \theta_{\mathrm{GJ}}, \phi_{\mathrm{GJ}}, m_r, \theta_{\mathrm{HF}}, \phi_{\mathrm{HF}}\}$, the graph will always be fed with six one-dimensional tensors with arbitrary length. So if we want to compute 1000 decay amplitudes we have to populate those six tensors with 1000 values each and feed them to the graph.

Chapter 3

Results

Now that we have implemented a TensorFlow graph which can compute decay amplitudes according to eq. (1.37) we can examine the algorithm in depth.

TensorFlow has APIs available in several programming languages both for constructing and executing a TensorFlow graph. The decay amplitude algorithm was implemented using the Python 2.7 and C++ APIs. The Python API was chosen because it is at present the most mature. Also Python is one of the most used programming languages by scientists today. The C++ API was chosen because ROOTPWA is almost entirely written in C++ and we aim to integrate TensorFlow into ROOTPWA. TensorFlow is being developed rather rapidly. The API version used for all following computations was 1.9. It should also be noted that all TensorFlow APIs other than the Python API are considered to be experimental by the developers and are not yet covered by the TensorFlow stability promises. All computations have been performed on a workstation of the CIP-Pool of the TUM Physikdepartment[1]. The source code is available at [13].

According to eq. (1.37), the computation of the decay amplitude needs six input values[2], i.e. two masses and four angles. Once the computation is completed, we obtain for each event two values, the real and imaginary part of the decay amplitude. The input data set consisted of 10 000 unique events, i.e. 10 000 times eight numerical values formatted in a CSV file[3]. There were eight values per event because the data set also provided the values of the real and imaginary part of the decay amplitude computed with ROOTPWA. Furthermore, in some tests we used up to 10^6 events. This larger data set was generated by copying the initial data set 100 times and merging all copies into a single file.

[1] To be precise, they were performed on workstation cont1sandy3, which possess a Intel i7 -2600K 3.40GHz (SandyBridge) CPU. More information about this machine can be found at ref. [12]

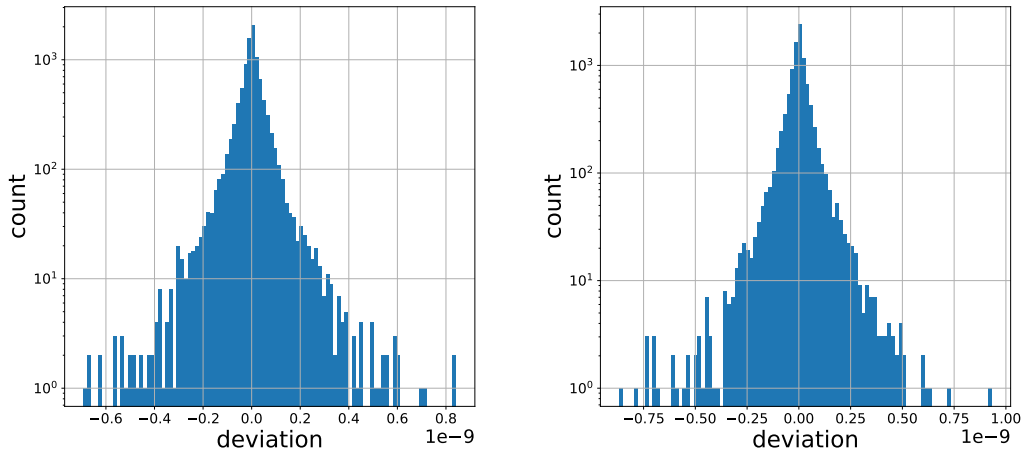
[2] All other values like quantum numbers, nominal masses and widths of the particles are constants of the computation and are hard coded into the TensorFlow graph itself.

[3] A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values.

3.1 Accuracy

In this section, we want to analyse how accurate our implemented algorithm can reproduce the results given by ROOTPWA[4]. Our algorithm and ROOTPWA are working with 64 bit double-precision floating-point numbers. The computation of the decay amplitude involves a large number of arithmetic operations. Since the order of execution of those operations will be different in the two implementations, we expect numerical differences of the calculated amplitude values due to different rounding effects.

The histograms of the differences of the real and imaginary part of the decay amplitudes as computed by the TensorFlow C++ implementation and ROOTPWA are shown in figs. 3.1a and 3.1b for all 10 000 events.



(a) Real part

(b) Imaginary Part

Figure 3.1: Difference between decay amplitudes as computed TensorFlow C++ implementation and ROOTPWA.

The histograms clearly show that most of the amplitude values are nearly identical. We also see that larger deviations are approximately exponentially suppressed. A good measure for the accuracy is the Root-Mean-Square-Deviation

[4]We will only use the output of the TensorFlow C++ implementation for this comparison. The results for the TensorFlow Python are nearly identical so there is no point in showing both.

(RMSD), which is defined by

$$\text{RMSD} = \sqrt{\frac{1}{n}\sum_i x_i^2} \tag{3.1}$$

where x_i are absolute differences between the real and imaginary parts respectively of two computed amplitudes. If we apply eq. (3.1) to the computed differences, we get for both the real and imaginary part an RMSD value of the order 10^{-10}. Compared with the uncertainty of the measured data, these deviations can be safely neglected. Therefore, we have shown that our algorithm reliably computes the same decay amplitudes as ROOTPWA.

3.2 Computation Time

Analysing the computation time of algorithms is a very broad field in computer science and it can be done in many different ways and with the use of advanced software tools. For the of sake of simplicity, we will not be using any profiling software. The easiest solution to measuring the execution time of any algorithm is by placing time stamps in the source code in such a way that the difference between two time stamps gives us the execution time of the operations between those two points.

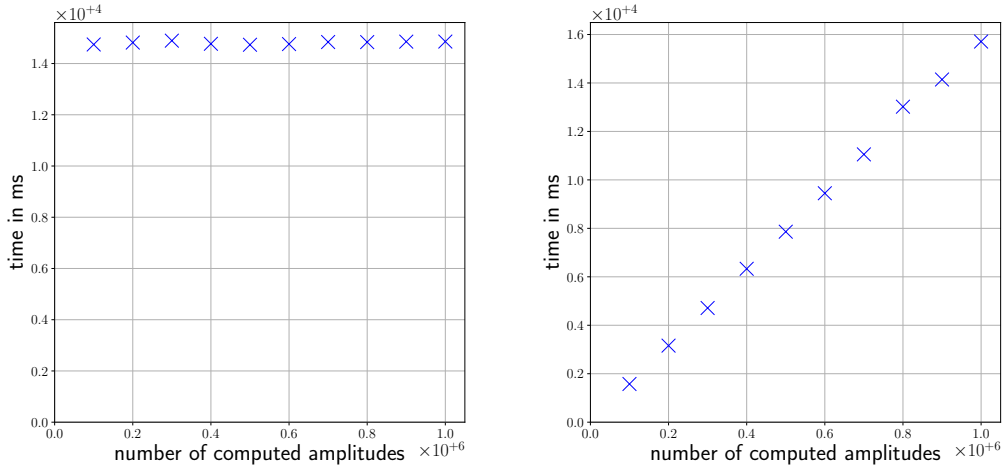
We will split the computation of the decay amplitudes into three parts. The first part of the computation is the reading of the input data into the TensorFlow graph and the writing of the computed amplitudes values into an output file. The second part is the construction[5] of the TensorFlow graph. Lastly, the most interesting part is the graph execution, i.e. the computation of the decay amplitudes. For every part, we compare the computation times of the Python and the C++ TensorFlow implementation.

3.2.1 Input and Output of Data

We will now analyse, how fast our algorithm can read in and write out data. The result for reading in data can be seen in fig. 3.2 and for writing data out in fig. 3.3.

The obvious difference between the two graphs in fig. 3.2 is that the time required by the Python implementation to read the data is virtually independent of the number of computed data points whereas the C++ implementation exhibits the expected linear dependence. This is due to how the algorithm was implemented

[5]The term "construction" refers to the compilation and the loading of the coded graph into the memory.



(a) Python implementation

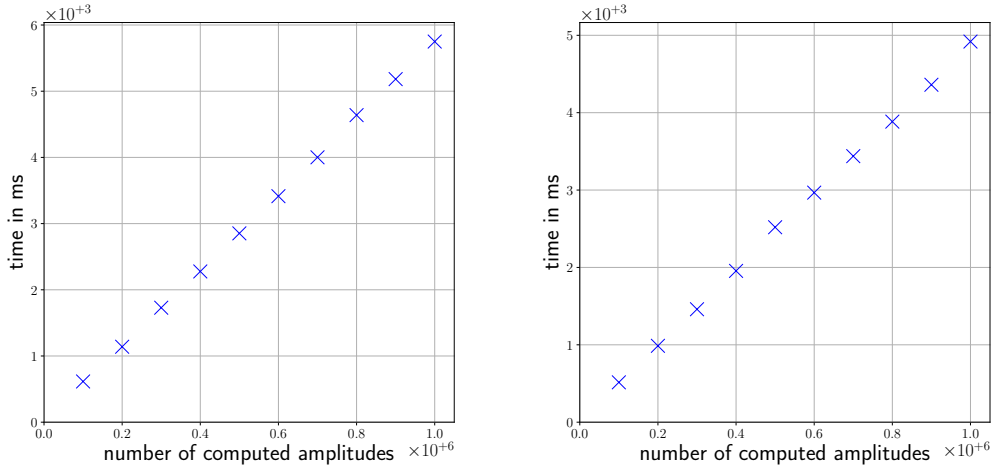
(b) C++ implementation

Figure 3.2: Time required to read data into the TensorFlow graph plotted against the number of computed amplitudes.

and how Python and C++ handle input files. Since C++ operates with so-called input/output streams, one can read small amounts of data from a large file at a sufficient speed. This explains the linear dependence of the time on the amount of data points in fig. 3.2b. In the Python implementation, we use a function from the NumPy package in order to read CSV files into arrays, which are then fed to the TensorFlow graph. However, this function always reads the whole file, i.e. all 10^6 events, even though not all of them were used during the computation. But when we look at the last data point for 10^6 computed amplitude values, we see that both implementations roughly require the same amount of time to read in the data. This is not surprising considering that NumPy is largely written in C. Hence, the different behaviour of the Python implementation is merely a consequence of how the test was implemented and does not indicate a performance issue of Python itself.

This discrepancy could possibly be avoided if we had used a TensorFlow mechanic to construct a so-called input pipeline. It was not realized in our case because the construction of such a pipeline is rather sophisticated and very specific to TensorFlow and this thesis mainly focuses on the actual computation. This is something which would be improved upon in further development.

The time required to write the computed amplitudes into a file, as shown in fig. 3.3, scales in both cases with the expected linear dependency. We also see that both require roughly the same amount of time. Since the same amount of data was



(a) Python implementation

(b) C++ implementation

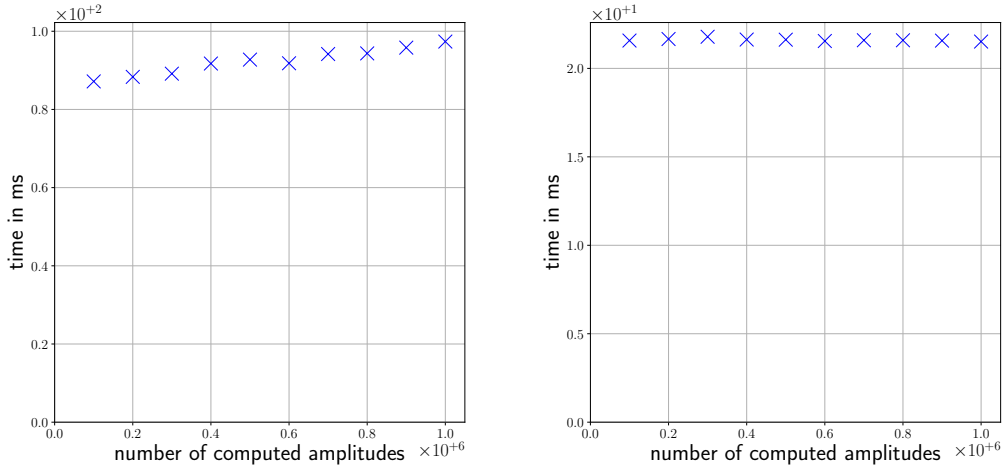
Figure 3.3: Time required to write data out of the TensorFlow graph plotted against the number of computed amplitudes.

fed to the TensorFlow graph in both implementations the same amount of data was computed and written to the output file. Hence, as explained above, we hardly see a difference, with regards to writing speed, between the two implementations.

3.2.2 Construction of the TensorFlow Graph

We will now examine the graph construction time. As noted above, graph construction refers to the process of compiling the coded TensorFlow graph and loading it into memory, so that it is ready for execution. The results are shown in fig. 3.4.

We can see that the construction time of the graph is independent of the amount of amplitudes we want to compute. This is because the graph is a representation of the mathematical of the decay amplitude, which is independent on the size of the input. The graph construction in the C++ implementation is about five times faster than in the Python implementation. In practice, however, the difference would probably be barely noticeable because the graph is only constructed once at the beginning of the computation and it does not make a big difference if the construction lasts 20 ms or 100 ms.



(a) Python implementation

(b) C++ implementation

Figure 3.4: Time required to construct the TensorFlow graph plotted against the number of computed amplitudes

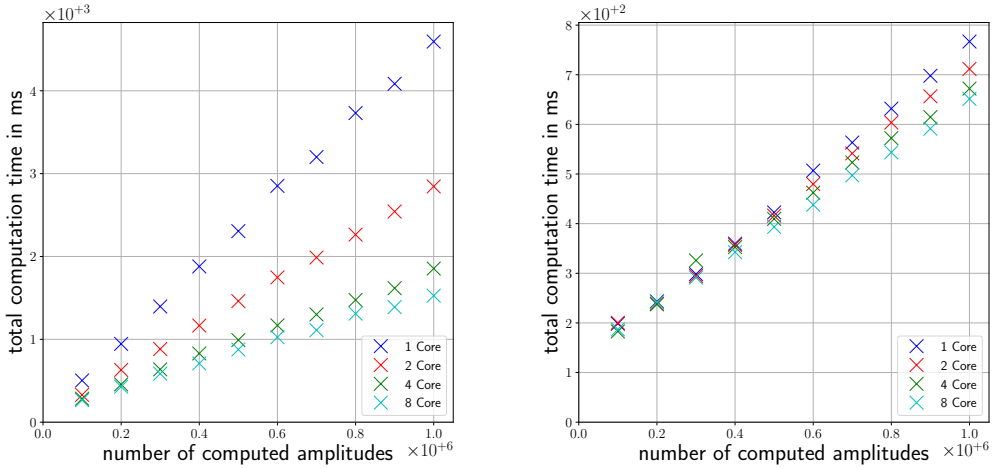
3.2.3 Graph Execution

We will now examine the time spent for the graph execution. Unlike for the parts of the computation we looked at before, we can now make use of the parallelization features provided by TensorFlow. As was already mentioned, using TensorFlow requires the implementation of the mathematical logic of the algorithm in form of a TensorFlow graph. To execute such a graph, a so-called TensorFlow session has to be created which is used to compute any node of the graph as long as we provide input. Such a TensorFlow session can be configured easily.

Since we are limiting ourselves to the computation on a single workstation, we will configure the number of CPU cores available to the TensorFlow session and look at how this impacts the computation time. The used workstation possesses eight CPU cores[6], so we analyse the performance of one, two, four and all eight cores. In addition to studying the computation time we will now also look at the average time required to compute a single decay amplitude in dependence on the number of computed amplitudes. This means we are normalizing the computation time to the number of computations. The results are shown in fig. 3.5 and fig. 3.6.

In fig. 3.5 we can see that, as expected, the total computation scales linearly with

[6]The CPU actually only has four physical CPU cores, but hyper-threading allows us to use eight virtual CPU cores.



(a) Python implementation

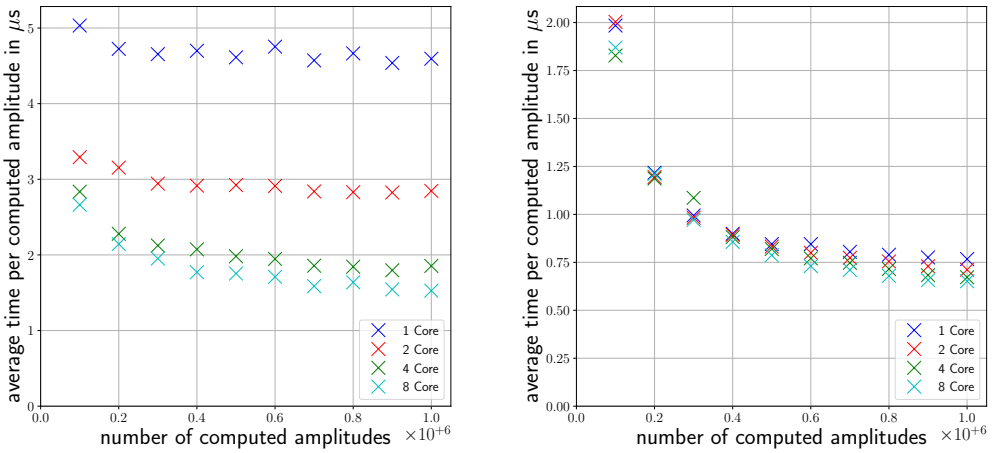
(b) C++ implementation

Figure 3.5: Computation time plotted against the number of computed amplitudes. Different colors represent different number of used CPU cores.

the number of computations and that if we supply more cores to the computation it takes less time. However, the speed-up does not scale linearly with the number of CPU cores and it becomes smaller the more CPUs are provided. In particular the improvement from four to eight is very small, but this is due to hyper-threading since the CPU only has four physical cores. Therefore, we will set the performance of four cores at 10^6 computed decay amplitudes as a benchmark and will compare it with the performance of one core.

The performance gain is large for the Python implementation as compared to the C++ implementation. If we compare the performance of one core to four cores for 10^6 computed amplitudes: in the Python implementation we gain roughly 274 ms, which is a relative improvement by a factor of 2.5. In the same instance, the C++ implementation improves by $9,4$ ms, which is a relative improvement by a factor of 1.1. We also see that the C++ implementation is faster than the Python implementation by 118 ms, or by a factor of 2.8.

In fig. 3.6 we can see that the average time needed to compute a single amplitude converges to a constant value in the limit of large data sets. The asymptotic value depends on the number of used CPUs. While this dependency is pronounced in case of the Python implementation, we gain $2,74\,\mu$s between one and four core performance at 10^6 amplitudes, the C++ implementation gains just $0.10\,\mu$s. The relative factors are the same as above.



(a) Python implementation

(b) C++ implementation

Figure 3.6: Time required for the computation of one decay amplitude plotted against the number of computed amplitudes. Different colors represent different number of used CPU cores.

By and large, we have found out that the C++ implementation is 2.8 times faster than the Python implementation at our set benchmark. The speed up is even larger for just one core. This is significant and very rather unexpected that the Python and C++ APIs differ so much in their computation prowess. Another unexpected result is the scaling behaviour of the TensorFlow C++ implementation with the number of used CPU cores. Usually one would expect two cut the computation by a factor close to two when doubling the number of CPU cores. While the Python implementation approximately displays this behaviour, the C++ implementation the computation time hardly changes. Furthermore, if we extrapolate the computation time of the C++ implementation in fig. 3.5b to zero computed amplitudes we see that there is a rather significant off-set. This is an lower limit to the achievable performance for small data sets. Without the use of profiling software it is very difficult to explain why these effects occur.

Chapter 4

Conclusions and Outlook

The main goal of this thesis was to proof that the computation of partial-wave decay amplitudes is feasible with the TensorFlow framework and that TensorFlow allows the computation to be parallelized on a single machine. As was shown in chapter 3, both goals were achieved. Another observation is that the C++ implementation seems to be more than 2.8 times faster than the Python implementation with regard to graph execution time. However, the C++ implementation does not scale well with the number of CPU cores used in parallel.

A next step would be to optimize the computations further based an profiling. Another step would be to implement graph optimization algorithms as discussed in section 2.1[1].

Looking at the big picture, now that we know how to deploy TensorFlow at a single workstation (CPU), in the future it should be investigated how to transfer the computation onto GPUs[2] and computer cluster.

[1] The chosen decay amplitude does not have much improvement potential in this regard so it was not included in this report. However, for other n-body final states, this may become important.
[2] The use of TensorFlow in conjunction with GPUs has been successfully tested already.

Appendix A

Expanded TensorFlow graphs

The following figures are the expanded TensorFlow subgraphs for the dynamical and angular displayed in fig. 2.2.

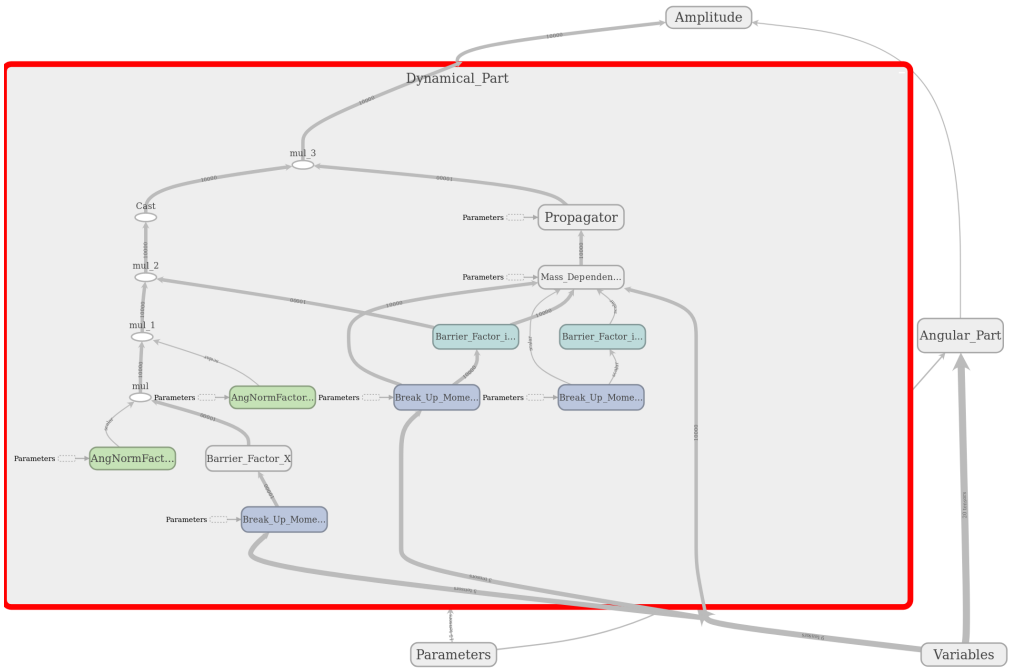


Figure A.1: TensorFlow graph representation of eq. (1.37) with expanded subgraph Dynamical_Part

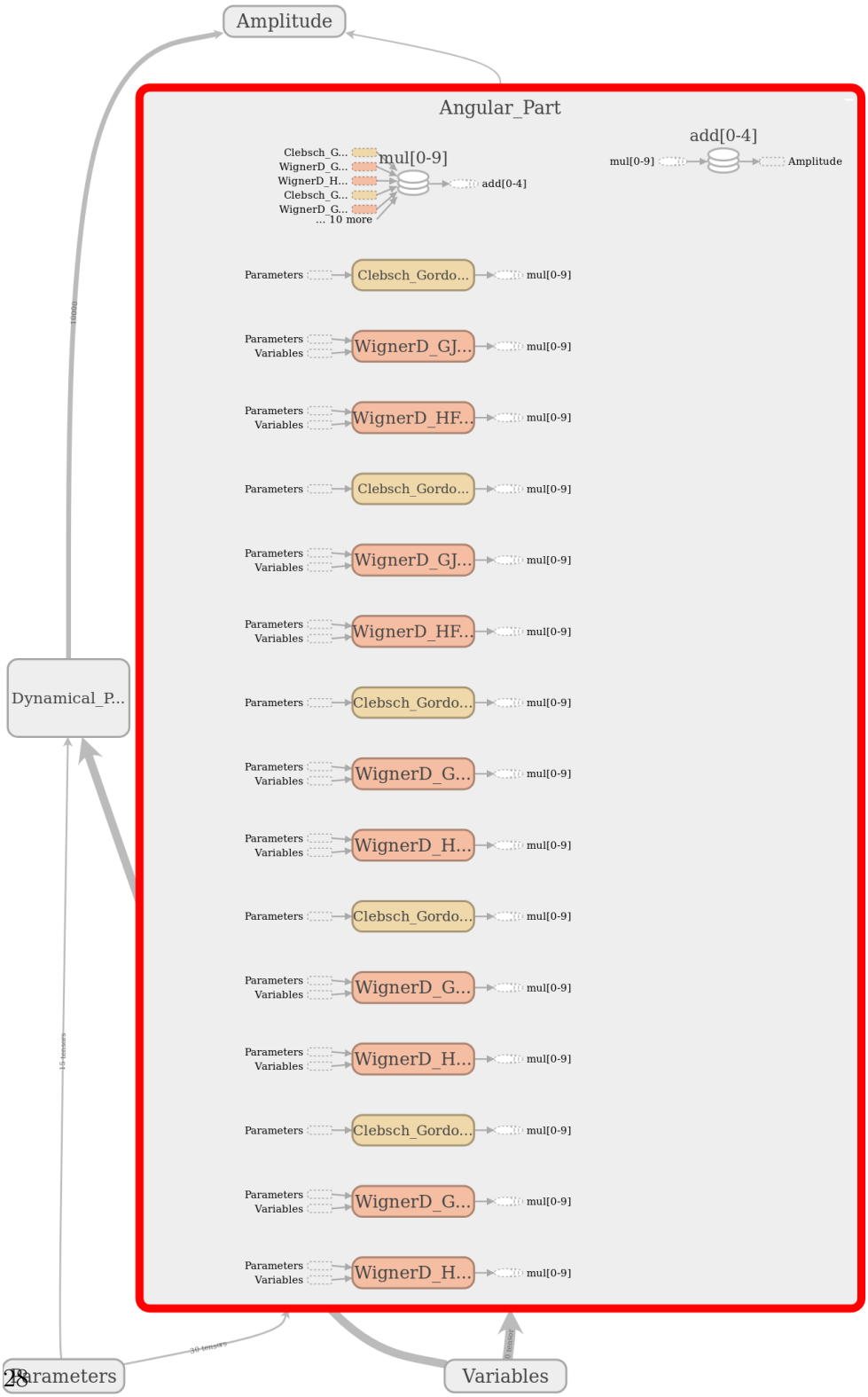


Figure A.2: TensorFlow graph representation of eq. (1.37) with expanded subgraph Angular_Part

List of Figures

Bibliography

[1] C. Adolph et al. »Resonance production and $\pi\pi$ S-wave in $\pi^- + p \to \pi^-\pi^-\pi^+ + p_{\text{recoil}}$ at 190 GeV/c«. In: *Phys. Rev. D* 95 (3 2017), p. 032004. DOI: 10.1103/PhysRevD.95.032004.

[2] J. D. Jackson and D. R. Tovey. *Kinematics*. 2017. URL: http://pdg.lbl.gov/2018/reviews/rpp2018-rev-kinematics.pdf.

[3] J. D. Hansen et al. »Formalism and assumptions involved in partial-wave analysis of three- meson systems«. In: *Nucl. Phys. B* 81.3 (1974), p. 403. DOI: 10.1016/0550-3213(74)90241-7.

[4] D. J. Herndon, P. Söding and R. J. Cashmore. »Generalized isobar model formalism«. In: *Phys. Rev. D* 11 (1975), p. 3165. DOI: 10.1103/PhysRevD.11.3165.

[5] Suh Urk Chung. *Spin formalisms; 3nd. updated version*. Upton, NY: Brookhaven Nat. Lab., 2013. URL: https://cds.cern.ch/record/1561144.

[6] A.D. Martin and T.D. Spearman. *Elementary particle theory*. North-Holland Pub. Co., 1970.

[7] E.P. Wigner and H.S.W. Massey. *Group Theory: And Its Application to the Quantum Mechanics of Atomic Spectra*. Elsevier Science, 2013.

[8] M.E. Rose. *Elementary Theory of Angular Momentum*. Dover books on physics and chemistry. Dover, 1995.

[9] J.M. Blatt and V.F. Weisskopf. *Theoretical Nuclear Physics*. Dover Books on Physics. Dover Publications, 2012.

[10] F. von Hippel and C. Quigg. »Centrifugal-Barrier Effects in Resonance Partial Decay Widths, Shapes, and Production Amplitudes«. In: *Phys. Rev. D* 5 (3 1972), pp. 624–638. DOI: 10.1103/PhysRevD.5.624.

[11] Michael Nielsen. *Neural Networks and Deep Learning*. Online; accessed 30-August-2018. 2017. URL: http://neuralnetworksanddeeplearning.com.

[12] Stefan Recksiegel. *Workstations*. Online; accessed 30-August-2018. 2017. URL: https://wiki.tum.de/display/tuphcip/Workstations.

[13] Anton Riedel. *E18/tensorFlowAmplitude*. Online; accessed 30-August-2018. 2018. URL: https://github.com/E18/tensorFlowAmplitude.