

$$e^+e^- \rightarrow \mu^+\mu^-$$

Alexander Huss

July 18, 2023

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Cross section and forward–backward asymmetry</b>	<b>1</b>
2.1	The squared Matrix Element . . . . .	1
2.1.1	Implementation . . . . .	2
2.2	Differential and total cross sections and $A_{FB}$ . . . . .	3
2.2.1	Implementation . . . . .	3
<b>3</b>	<b>Playground</b>	<b>3</b>
3.1	Export source code . . . . .	3
3.2	Collider energy scan . . . . .	4
3.3	$M_Z$ variation . . . . .	6
3.4	$\Gamma_Z$ variation . . . . .	6
3.5	$A_{FB}$ and the weak mixing angle . . . . .	7

## 1 Introduction

We will implement the process  $e^+e^- \rightarrow \mu^+\mu^-$  at leading order. This will give us an idea of how predictions are done in practice.

## 2 Cross section and forward–backward asymmetry

### 2.1 The squared Matrix Element

We have seen in the lecture that the squared amplitude (summed/averaged over final-/initial-state degrees of freedom) is given by

$$\frac{1}{4} \sum_{\text{spins}} |\mathcal{M}_\gamma + \mathcal{M}_Z|^2 = e^4 \left[ G_1(s) (1 + \cos^2(\theta)) + G_2(s) 2 \cos(\theta) \right] \quad (1)$$

with the functions

$$G_1(s) = 1 + 2v_e v_\mu \operatorname{Re} \left\{ \frac{s}{s - M_Z^2 + i\Gamma_Z M_Z} \right\} + (v_e^2 + a_e^2)(v_\mu^2 + a_\mu^2) \left| \frac{s}{s - M_Z^2 + i\Gamma_Z M_Z} \right|^2$$

$$G_2(s) = 0 + 2a_e a_\mu \operatorname{Re} \left\{ \frac{s}{s - M_Z^2 + i\Gamma_Z M_Z} \right\} + 4v_e a_e v_\mu a_\mu \left| \frac{s}{s - M_Z^2 + i\Gamma_Z M_Z} \right|^2$$

### 2.1.1 Implementation

We'll use a simple class to save and retrieve Standard Model parameters including some convenience functions:

---

```
class Parameters(object):
    """very simple class to manage Standard Model Parameters"""

    #> conversion factor from GeV^{-2} into nanobarns [nb]
    GeVnb = 0.389e6;

    def __init__(self, **kwargs):
        #> these are the independent variables we chose:
        #> * sw2 = sin^2(theta_w) with the weak mixing angle theta_w
        #> * (MZ, GZ) = mass & width of Z-boson
        self.sw2 = kwargs.pop("sw2", 0.223)
        self.MZ = kwargs.pop("MZ", 91.1876)
        self.GZ = kwargs.pop("GZ", 2.4952)
        if len(kwargs) > 0:
            raise RuntimeError("passed unknown parameters: {}".format(kwargs))
        #> let's store some more constants (l, u, d = lepton, up-quark, down-quark)
        self.Ql = -1.; self.I3l = -1./2.;
        self.Qu = +2./3.; self.I3u = +1./2.;
        self.Qd = -1./3.; self.I3d = -1./2.;
        self.alpha = 1./137.
        #> and some derived quantities
        self.sw = math.sqrt(self.sw2)
        self.cw2 = 1.-self.sw2 # cos^2 = 1-sin^2
        self.cw = math.sqrt(self.cw2)
        #> vector & axial-vector couplings to Z-boson
        @property
        def vl(self) -> float:
            return (self.I3l-2*self.Ql*self.sw2)/(2.*self.sw*self.cw)
        @property
        def al(self) -> float:
            return self.I3l/(2.*self.sw*self.cw)
        #> the Z-boson propagator
        def propZ(self, s: float) -> complex:
            return s/(s-complex(self.MZ**2,self.GZ*self.MZ))
    #> we immediately instantiate an object (default values) in global scope
    PARAM = Parameters()
```

---

We next implement the functions  $G_1$  and  $G_2$  that were introduced to express the squared Matrix Element in terms of the even and odd components w.r.t.  $\cos(\theta)$ :

---

```
def G1(s: float, par=PARAM) -> float:
    return par.Ql**2 - 2. * par.vl**2 * par.Ql * par.propZ(s).real + (par.vl**2 + par.al**2)**2 * abs(par.propZ(s))**2
def G2(s: float, par=PARAM) -> float:
    return -2. * par.al**2 * par.Ql * par.propZ(s).real + 4. * par.vl**2 * par.al**2 * abs(par.propZ(s))**2
```

---

## 2.2 Differential and total cross sections and $A_{FB}$

The formula for the differential cross section reads

$$\frac{d\sigma}{d\cos\theta} = \frac{\alpha^2\pi}{2s} \left[ G_1(s) (1 + \cos^2(\theta)) + G_2(s) 2\cos(\theta) \right], \quad (2)$$

where  $s = (p_{e+} + p_{e-})^2 = (p_{\mu+} + p_{\mu-})^2 = 4E_B^2$  is the centre-of-mass energy of the collision and  $\theta$  the scattering angle between the electron and the muon.

We obtain the total cross section by integrating over  $\cos\theta$ :

$$\sigma = \int_{-1}^{+1} d\cos\theta \frac{d\sigma}{d\cos\theta} = \frac{\alpha^2\pi}{2s} \frac{8}{3} G_1(s). \quad (3)$$

Another interesting quantity to look at is the forward-backward asymmetry defined as:

$$A_{FB} = \frac{1}{\sigma} \left\{ \int_0^{+1} d\cos\theta \frac{d\sigma}{d\cos\theta} - \int_{-1}^0 d\cos\theta \frac{d\sigma}{d\cos\theta} \right\} = \frac{3}{4} \frac{G_2(s)}{G_1(s)}. \quad (4)$$

### 2.2.1 Implementation

We'll start with the implementation of the total cross section

---

```
def cross(s: float, par=PARAM) -> float:
    return par.GeVnb * par.alpha**2*math.pi/(2.*s) * (8./3.) * G1(s, par)
```

---

and define another function for the forward-backward asymmetry

---

```
def AFB(s: float, par=PARAM) -> float:
    return (3./4.) * G2(s,par)/G1(s,par)
```

---

## 3 Playground

### 3.1 Export source code

We can export the python source code to a file `main.py`

---

```
import math
import cmath
import numpy as np
class Parameters(object):
    """very simple class to manage Standard Model Parameters"""

    #> conversion factor from GeV^{-2} into nanobarns [nb]
    GeVnb = 0.389e6;

    def __init__(self, **kwargs):
        #> these are the independent variables we chose:
        #> * sw2 = sin^2(theta_w) with the weak mixing angle theta_w
        #> * (MZ, GZ) = mass & width of Z-boson
```

```

self.sw2 = kwargs.pop("sw2", 0.223)
self.MZ = kwargs.pop("MZ", 91.1876)
self.GZ = kwargs.pop("GZ", 2.4952)
if len(kwargs) > 0:
    raise RuntimeError("passed unknown parameters: {}".format(kwargs))
#> let's store some more constants (l, u, d = lepton, up-quark, down-quark)
self.Ql = -1.; self.I3l = -1./2.;
self.Qu = +2./3.; self.I3u = +1./2.;
self.Qd = -1./3.; self.I3d = -1./2.;
self.alpha = 1./137.
#> and some derived quantities
self.sw = math.sqrt(self.sw2)
self.cw2 = 1.-self.sw2 # cos^2 = 1-sin^2
self.cw = math.sqrt(self.cw2)
#> vector & axial-vector couplings to Z-boson
@property
def vl(self) -> float:
    return (self.I3l-2*self.Ql*self.sw2)/(2.*self.sw*self.cw)
@property
def al(self) -> float:
    return self.I3l/(2.*self.sw*self.cw)
#> the Z-boson propagator
def propZ(self, s: float) -> complex:
    return s/(s-complex(self.MZ**2,self.GZ*self.MZ))
#> we immediately instantiate an object (default values) in global scope
PARAM = Parameters()
def G1(s: float, par=PARAM) -> float:
    return par.Ql**2 - 2. * par.vl**2 * par.Ql * par.propZ(s).real + (par.vl**2 + par.al**2)**2 * abs(par.propZ(s))**2
def G2(s: float, par=PARAM) -> float:
    return -2. * par.al**2 * par.Ql * par.propZ(s).real + 4. * par.vl**2 * par.al**2 * abs(par.propZ(s))**2
def cross(s: float, par=PARAM) -> float:
    return par.GeVnb * par.alpha**2*math.pi/(2.*s) * (8./3.) * G1(s, par)
def AFB(s: float, par=PARAM) -> float:
    return (3./4.) * G2(s,par)/G1(s,par)
if __name__ == "__main__":
    res = []
    for Ecm in np.linspace(20, 100, 200):
        s = Ecm**2
        xs = cross(s)
        afb = AFB(s)
        print("{:e} {:e} {:e}".format(Ecm,xs,afb))

```

---

by using the `tangle` command

---

```
(org-babel-tangle)
```

---

You can play around with the source and execute it as usual

---

```
python main.py
```

---

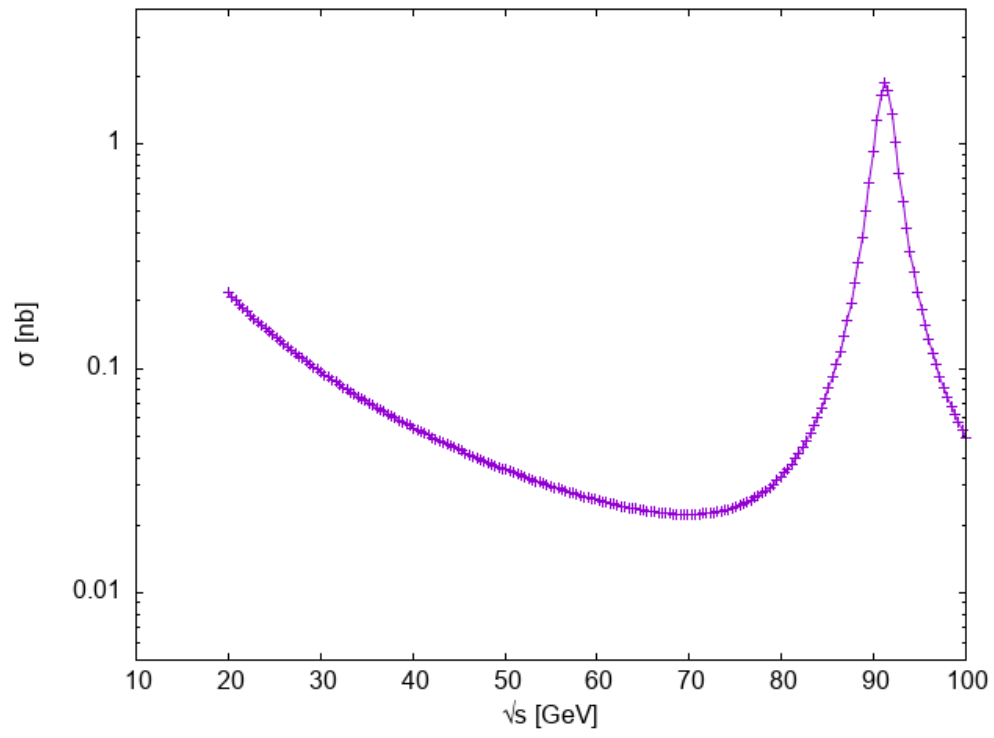
## 3.2 Collider energy scan

---

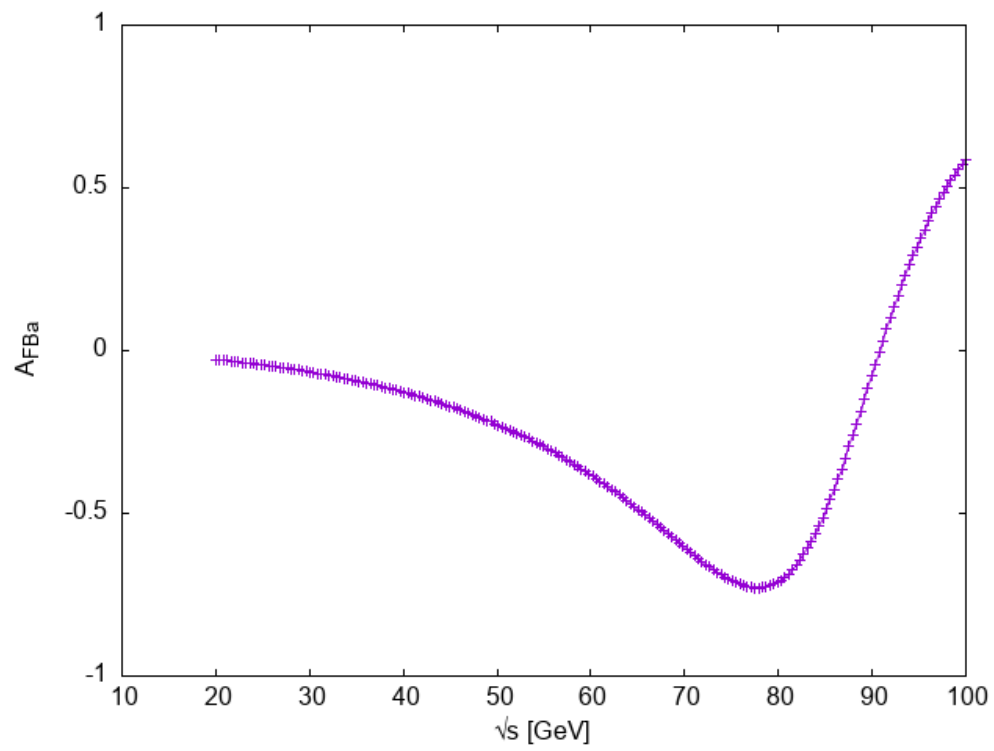
```
python main.py
```

---

The total cross section



The forward-backward asymmetry



### 3.3 $M_Z$ variation

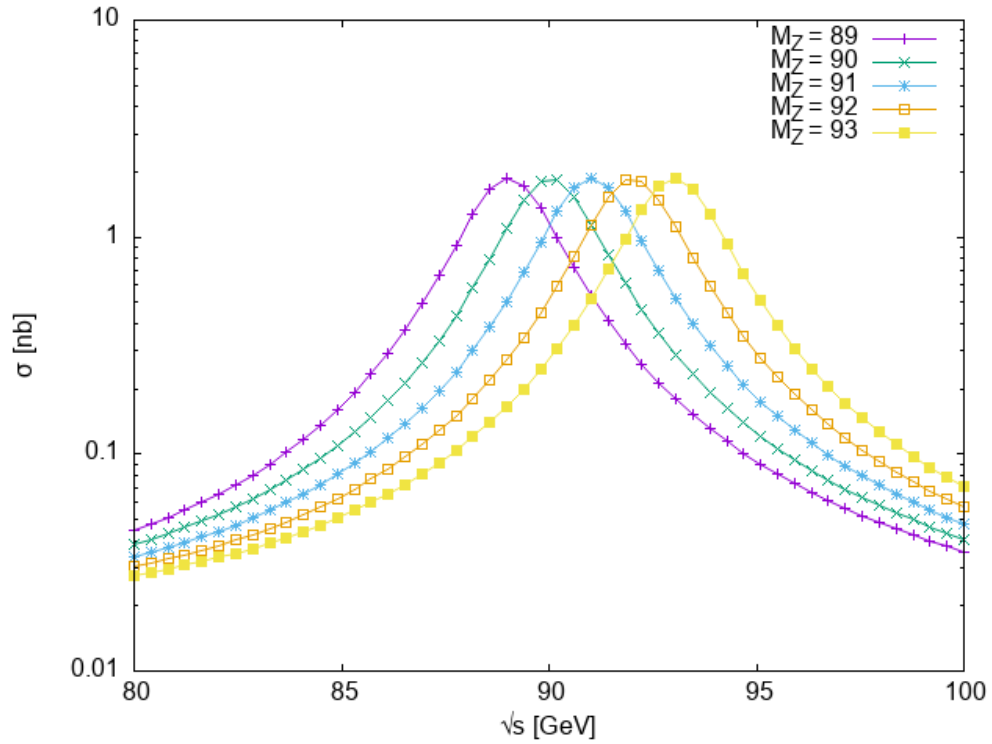
Let's see how the cross section behaves under variation of the Z-boson mass

---

```
import math
import cmath
import numpy as np
<<util>>
<<cross>>
res = []
MZ_scan = [ Parameters(MZ=val) for val in [89, 90, 91, 92, 93] ]
for Ecms in np.linspace(80, 100, 50):
    s = Ecms**2
    ires = [Ecms]
    for par in MZ_scan:
        xs = cross(s, par)
        ires.append(xs)
    res.append(ires)
return res
```

---

let's plot the dependence on the Z-boson mass around the resonance



### 3.4 $\Gamma_Z$ variation

Let's check how the picture would change if we had a different number of light neutrino species. The branching fraction of a Z-boson decay into neutrino ("invisible decay") is 20%.

---

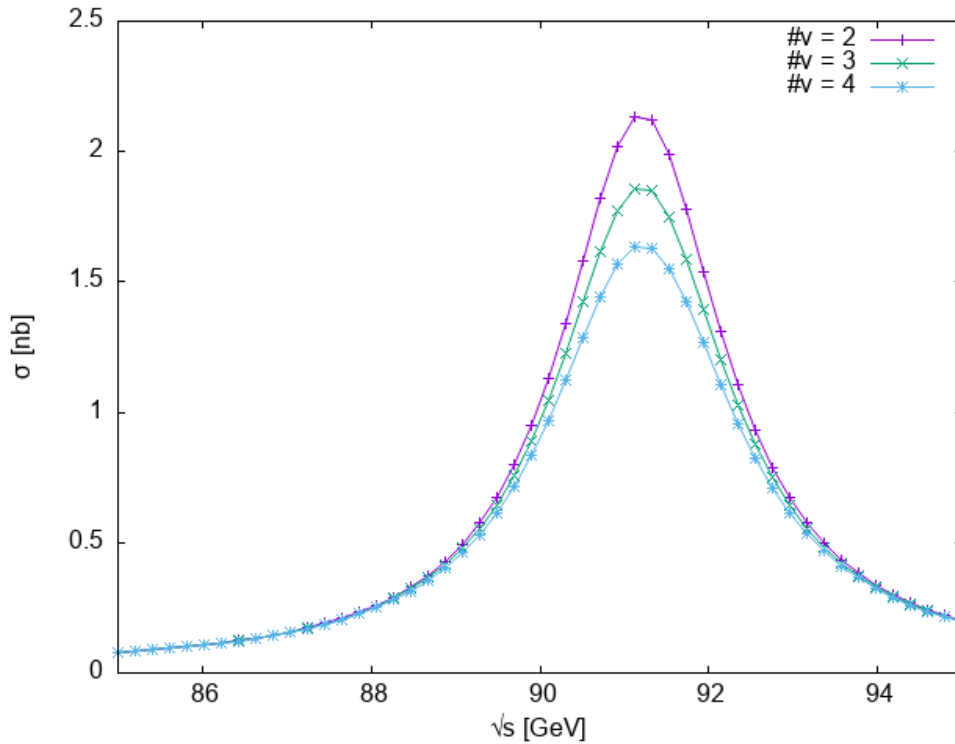
```

import math
import cmath
import numpy as np
<<util>>
<<cross>>
res = []
#> the partial decay width for Z -> massless (anti-)neutrino
GZ_nu = 0.2 * PARAM.GZ / 3.
GZ_scan = [ Parameters(GZ=PARAM.GZ-GZ_nu), PARAM, Parameters(GZ=PARAM.GZ+GZ_nu) ]
for Ecms in np.linspace(85, 95, 50):
    s = Ecms**2
    ires = [Ecms]
    for par in GZ_scan:
        xs = cross(s, par)
        ires.append(xs)
    res.append(ires)
return res

```

---

let's plot how much the Z line shape varies with the number of neutrino generations



### 3.5 $A_{FB}$ and the weak mixing angle

The forward-backward asymmetry is an observable that is sensitive to the weak mixing angle as we will see in the following. Moreover, defined as a ratio, many systematic uncertainties cancel.

---

```

import math
import cmath

```

```

import numpy as np
<<util>>
<<cross>>
res = []
#> the partial decay width for Z -> massless (anti-)neutrino
sw2_step = PARAM.sw2 * 0.1 # 10% variation per step
sw2_scan = [ Parameters(sw2=PARAM.sw2+i*sw2_step) for i in [-3,-2,-1,0,1,2,3] ]
for Ecms in np.linspace(85, 95, 50):
    s = Ecms**2
    ires = [Ecms]
    for par in sw2_scan:
        afb = AFB(s, par)
        ires.append(afb)
    res.append(ires)
return res

```

---

let's see how much  $A_{FB}$  varies with  $\sin^2 \theta_w$ :

