$e^+e^- \to \mu^+\mu^-$

Alexander Huss

July 20, 2025

Contents

1 Introduction

We will implement the process $e^+e^- \to \mu^+\mu^-$ at leading order. This is among the simplest processes there are but it gives us many knobs to play around with to get an idea about the physics underlying these predictions.

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

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

with the functions

$$G_1(s) = 1 + 2v_\mathrm{e}v_\mu \mathrm{Re}\left\{\frac{s}{s - M_\mathrm{Z}^2 + \mathrm{i}\Gamma_\mathrm{Z} M_\mathrm{Z}}\right\} + (v_\mathrm{e}^2 + a_\mathrm{e}^2)(v_\mu^2 + a_\mu^2)\left|\frac{s}{s - M_\mathrm{Z}^2 + \mathrm{i}\Gamma_\mathrm{Z} M_\mathrm{Z}}\right|^2$$

$$G_2(s) = 0 + 2a_\mathrm{e}a_\mu \mathrm{Re}\left\{\frac{s}{s - M_\mathrm{Z}^2 + \mathrm{i}\Gamma_\mathrm{Z} M_\mathrm{Z}}\right\} + 4v_\mathrm{e}a_\mathrm{e}v_\mu a_\mu\left|\frac{s}{s - M_\mathrm{Z}^2 + \mathrm{i}\Gamma_\mathrm{Z} M_\mathrm{Z}}\right|^2$$

2.1.1 Implementation

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

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

    #> conversion factor from GeV^{-2} into nanobarns [nb]
    GeVnb = 0.3893793656e6

    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.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. $\cos(\theta)$:

```python
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

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

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

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

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

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

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

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 = 0.3893793656e6

    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.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))
```

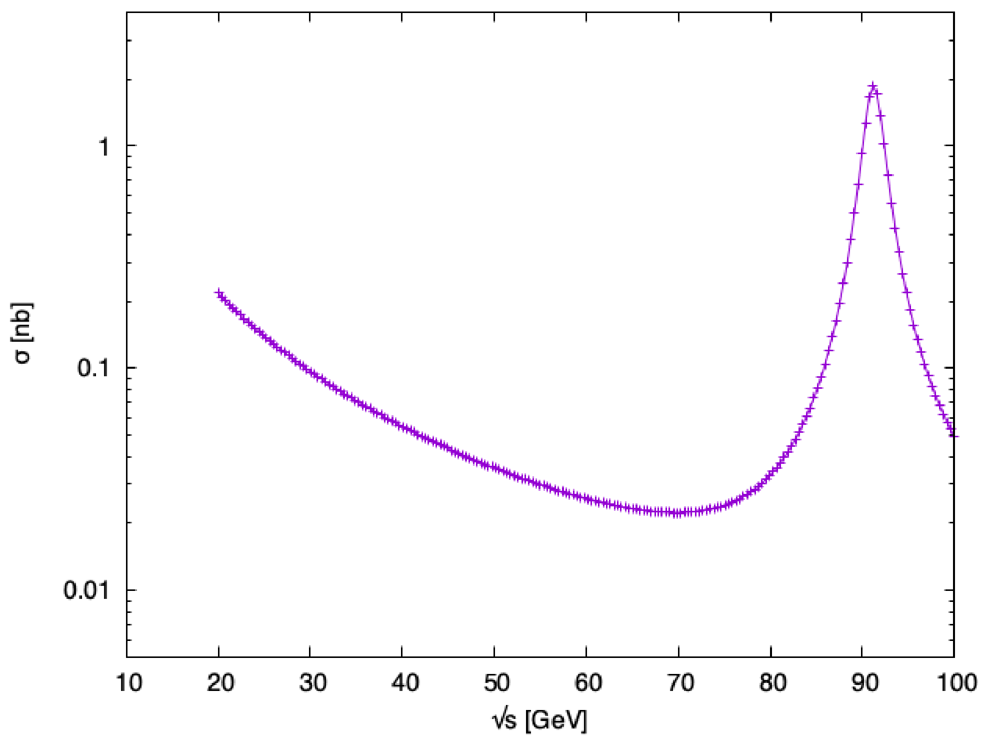
by using the `tangle` command

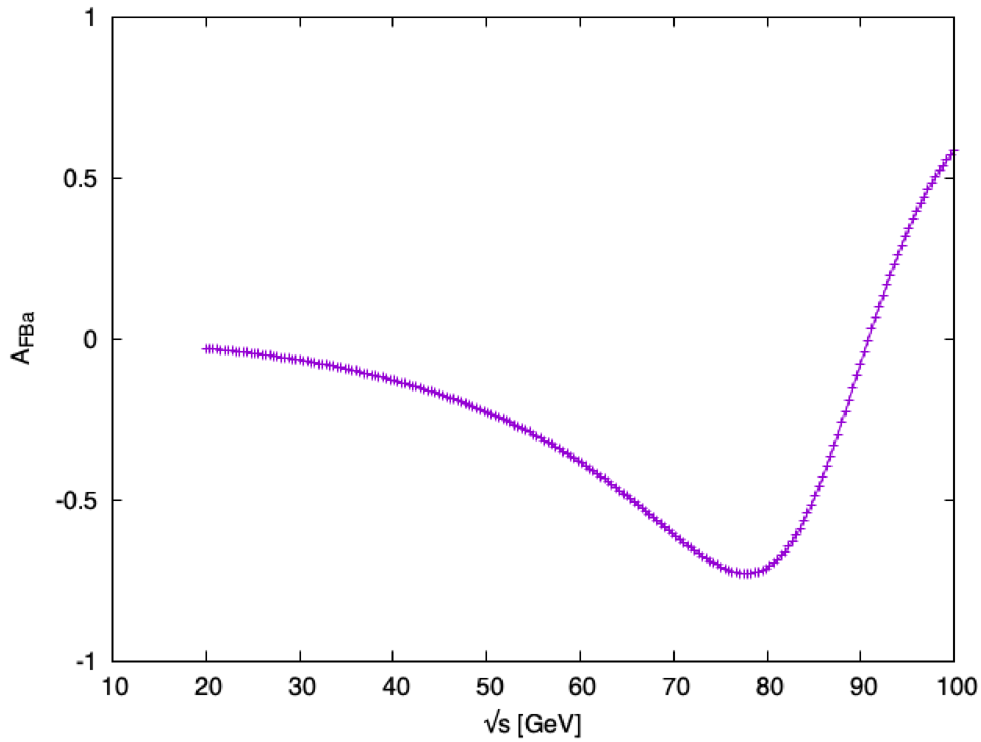
```
(org-babel-tangle)
```

3.2 Collider energy scan

Let's execute the python script we just exported and look at the total cross section and the forward–backward asymmetry as a function of the collider energy.

```
python main.py
```


σ [nb]
1
0.1
0.01
10
20
30
40
50
60
70
80
90
100
√s [GeV]

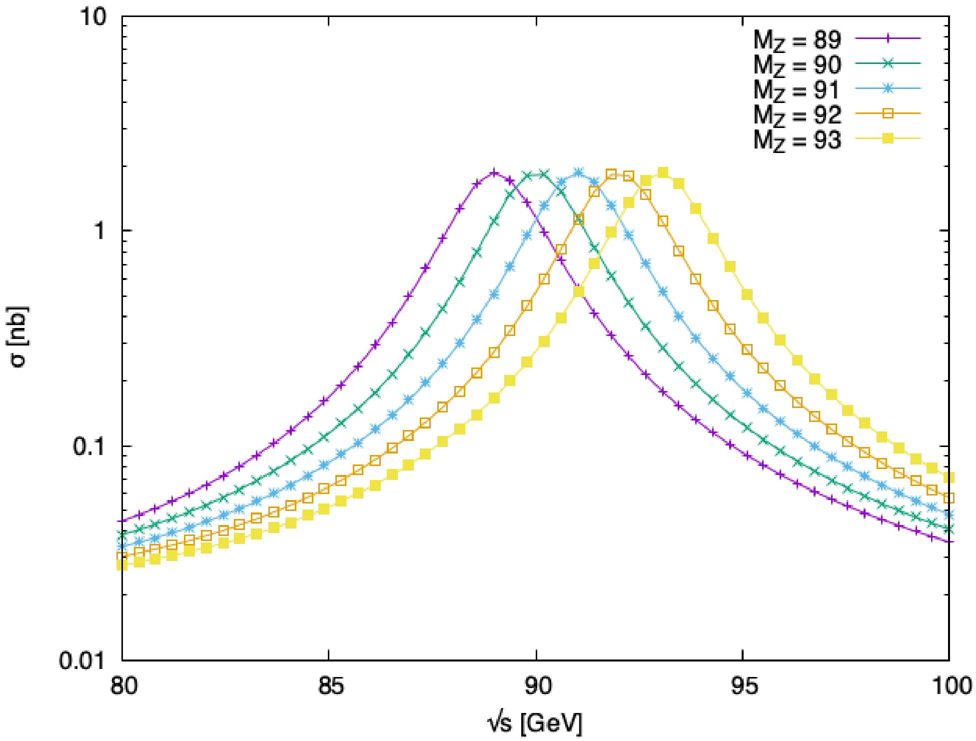

A_{FBa}
1
0.5
0
-0.5
-1
10
20
30
40
50
60
70
80
90
100
√s [GeV]

3.3 $M_{\rm Z}$ 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.item()]
    for par in MZ_scan:
        xs = cross(s, par)
        ires.append(xs.item())
    res.append(ires)
return res
```

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

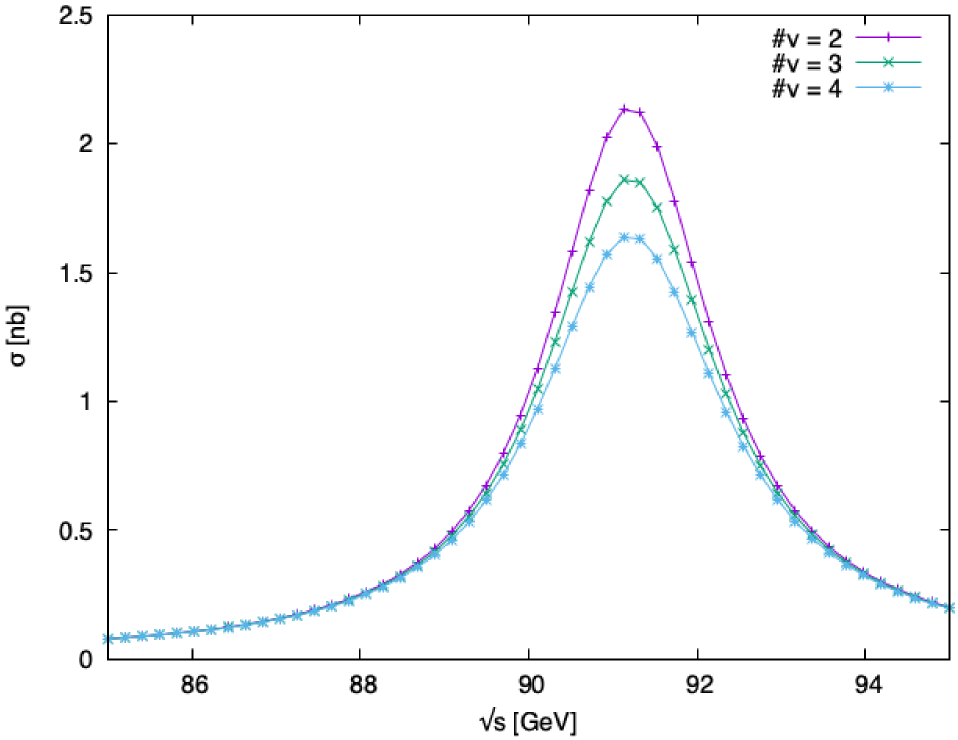


3.4 $\Gamma_{\rm Z}$ variation

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.item()]
    for par in GZ_scan:
        xs = cross(s, par)
        ires.append(xs.item())
    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.item()]
    for par in sw2_scan:
        afb = AFB(s, par)
        ires.append(afb.item())
    res.append(ires)
return res
```

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