Calculation of the differential cross section for $pp \to Z \to e^+e^-$ at the LHC with $\sqrt{s} = 7$ TeV

We will now use the code provided in Lecture 40 to weight the parton-level cross section for $q\bar{q} \to Z \to e^+e^-$ by the MSTW PDF sets at $\sqrt{s} = 7$ TeV for a pp collider. In this case, the protons have equal and opposite momenta:

$p_1 = (0, 0, -\sqrt{s}/2, \sqrt{s}/2)$

$p_2 = (0, 0, \sqrt{s}/2, \sqrt{s}/2)$

Each quark in the proton will have momenta

$p_1 = (0, 0, -x_1\sqrt{s}/2, x_1\sqrt{s}/2)$

$p_2 = (0, 0, x_2\sqrt{s}/2, x_2\sqrt{s}/2)$

where x_1 and x_2 are the momentum fractions of the quark.

This calculation will involve the calculation of the integral over all possible momentum fractions of the quarks ($0 \to 1$):

$$\frac{d\sigma}{d\cos(\theta)} = \sum_{a_1,a_2} \int_0^1 dx_1 dx_2 f_{a_1}(x_1, M^2) f_{a_2}(x_2, M^2) E_Q \frac{d\sigma_{a_1,a_2}}{d^3Q}(x_1 P_1, x_2 P_2, M^2)$$

We choose $M^2 = 100$ GeV, and we loop over quark species $a_i = 1, 2, 3$ where 1,2,3 are the ID numbers for down, up, and strange quarks, respectively. The functions f_{a_i} are the parton distribution functions from the MSTW PDF sets and are evaluated at momentum fractions x_i.

My strategy is to write the code in `C++` and use `VEGAS` to do the integration, which can be done in `python`. The technical trick is to create a class with an `operator()` method (i.e. a functor) that can implement the function call for each value of x_1 and x_2. Remember, however, that `VEGAS` expects a signature like

```
double ClassName::operator()( std::vector<double> const &
x)
```

so we have to pass our values of x_1 and x_2 appropriately. As such, the core of this code is implemented in CrossSection.cpp (CrossSection.cpp):

```
double dSigmaDOmega_PP::operator()( std::vector<double> co
nst & x )
{
  double pdf1 = pdf_.parton( flavor_, x[0], q_);
  double pdf2 = pdf_.parton(-flavor_, x[1], q_);

  Particle q_i      ( LorentzVector(0,0, e_com_*x[0]*0.5,
e_com_*x[0]*0.5),  flavor_);
  Particle qbar_i   ( LorentzVector(0,0,-e_com_*x[1]*0.5,
e_com_*x[1]*0.5), -flavor_);
  double e = (q_i.p4() + q_i.p4()).e();
  return pdf1*pdf2*e*dSigmaDOmega( q_i, qbar_i, 11, cosThe
ta_ );
}
```

The rest of the code is looping over flavors and $\cos\theta$ values, and letting `VEGAS` run the integration.

One important piece is to not let the algorithm sample values too close to $x = 0$ because it will introduce numerical instabilities. I pick 0.0001 as a minimum value, which would correspond to around 0.35 MeV of energy per parton. This is pretty low compared to the mass of the Z boson, which is 91.5 GeV.

VEGAS

The `VEGAS` implementation can be used in `python` directly. This involves the setting of several parameters: the limits of integration ($0.0001 \rightarrow 1.0$); the number of iterations to adjust the binning (3-5 is appropriate); and the number of MC steps (50k per integral for the production step is appropriate).

Correct calculation of Cross Section

The cross section will therefore need to loop over the appropriate quark species. Remember as in class, there are "valence" quarks (that contribute to the quantum numbers of the proton) and "sea" quarks (that do not contribute to the quantum numbers of the proton, since they always appear as $q\overline{q}$ pairs).

The valence quarks include two up quarks and one down quark. However, it is important to account also for the "sea" quarks. The relevant pieces are:

- Up quarks: $u_v + u_s$ combined with u_s.
- Down quarks: $d_v + d_s$ combined with d_s.
- Strange quarks: s_s twice.

The MSTW PDF sets also provide convenience functions that will account for this, so you can do either:

```
double pdf1 = pdf.parton( flavor, x1, q);
double pdf2 = pdf.parton(-flavor, x2, q);
```

or

```
    // Assume your quark species is "type" and is 1,2, or
3:
    pdf->update(x,q);
    double upv = pdf->cont.upv;
    double dnv = pdf->cont.dnv;
    double usea = pdf->cont.usea;
    double dsea = pdf->cont.dsea;
    double str = pdf->cont.str;
    double sbar = pdf->cont.sbar;
    double pdf1 = 0.0, pdf2 = 0.0;
    if ( type == 1){
        pdf1 = (dnv + dsea);
        pdf2 = dsea;
    } else if ( type == 2 ){
        pdf1 = (upv + usea);
        pdf2 = usea;
    } else if ( type == 3 ){
        pdf1 = str;
        pdf2 = sbar;
    }
```

Results

The example codes calculate the individual quark species (up,down,strange) separately so you can see the effects. You then add them up remembering that there are twice as many up quarks as down quarks.

In [1]:

```
! swig -c++ -python swig/qft.i
! python swig/setup_qft.py build_ext --inplace
```

```
running build_ext
building '_qft' extension
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Ws
ign-compare -DNDEBUG -g -fwrapv -O2 -Wall -g -fstack
-protector-strong -Wformat -Werror=format-security -
g -fwrapv -O2 -g -fstack-protector-strong -Wformat -
Werror=format-security -Wdate-time -D_FORTIFY_SOURCE
=2 -fPIC -I/usr/include/python3.7m -c swig/qft_wrap.
cxx -o build/temp.linux-x86_64-3.7/swig/qft_wrap.o -
I./ -std=c++11 -O3
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Ws
ign-compare -DNDEBUG -g -fwrapv -O2 -Wall -g -fstack
-protector-strong -Wformat -Werror=format-security -
g -fwrapv -O2 -g -fstack-protector-strong -Wformat -
Werror=format-security -Wdate-time -D_FORTIFY_SOURCE
=2 -fPIC -I/usr/include/python3.7m -c LorentzVector.
cpp -o build/temp.linux-x86_64-3.7/LorentzVector.o -
I./ -std=c++11 -O3
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Ws
ign-compare -DNDEBUG -g -fwrapv -O2 -Wall -g -fstack
-protector-strong -Wformat -Werror=format-security -
g -fwrapv -O2 -g -fstack-protector-strong -Wformat -
Werror=format-security -Wdate-time -D_FORTIFY_SOURCE
=2 -fPIC -I/usr/include/python3.7m -c Particle.cpp -
o build/temp.linux-x86_64-3.7/Particle.o -I./ -std=c
++11 -O3
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Ws
ign-compare -DNDEBUG -g -fwrapv -O2 -Wall -g -fstack
-protector-strong -Wformat -Werror=format-security -
g -fwrapv -O2 -g -fstack-protector-strong -Wformat -
Werror=format-security -Wdate-time -D_FORTIFY_SOURCE
=2 -fPIC -I/usr/include/python3.7m -c CrossSection.c
pp -o build/temp.linux-x86_64-3.7/CrossSection.o -I.
```

```
/ -std=c++11 -O3
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Ws
ign-compare -DNDEBUG -g -fwrapv -O2 -Wall -g -fstack
-protector-strong -Wformat -Werror=format-security -
g -fwrapv -O2 -g -fstack-protector-strong -Wformat -
Werror=format-security -Wdate-time -D_FORTIFY_SOURCE
=2 -fPIC -I/usr/include/python3.7m -c mstwpdf.cpp -o
build/temp.linux-x86_64-3.7/mstwpdf.o -I./ -std=c++1
1 -O3
x86_64-linux-gnu-g++ -pthread -shared -Wl,-O1 -Wl,-B
symbolic-functions -Wl,-Bsymbolic-functions -Wl,-z,r
elro -Wl,-Bsymbolic-functions -Wl,-z,relro -g -fstac
k-protector-strong -Wformat -Werror=format-security
-Wdate-time -D_FORTIFY_SOURCE=2 build/temp.linux-x86
_64-3.7/swig/qft_wrap.o build/temp.linux-x86_64-3.7/
LorentzVector.o build/temp.linux-x86_64-3.7/Particle
.o build/temp.linux-x86_64-3.7/CrossSection.o build/
temp.linux-x86_64-3.7/mstwpdf.o -o /results/physics-
assignment-2-rappoccio-1/Assignment2/_qft.cpython-37
m-x86_64-linux-gnu.so
```

In [2]:

```python
import sys
import os
sys.path.append( os.path.abspath("swig") )
import qft
import numpy as np
import matplotlib.pyplot as plt
import vegas
```

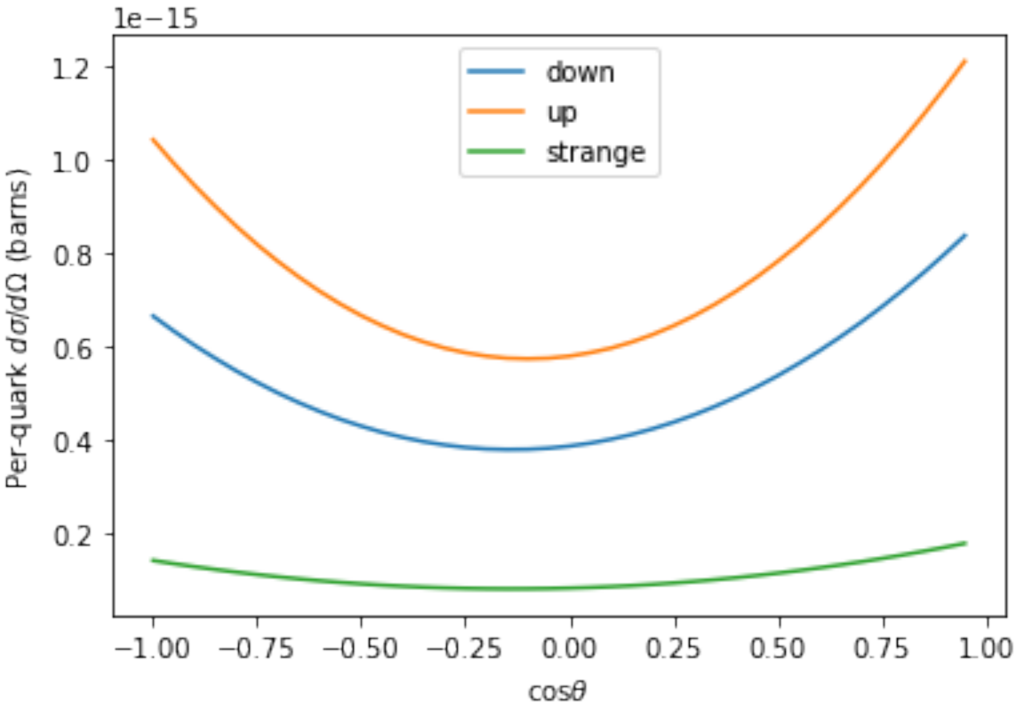
In [3]:

```
regn = [ [0.0001, 1.0], # x1
         [0.0001, 1.0]] # x2

grid_file_name = "Grids/mstw2008lo.00.dat"
pdf = qft.c_mstwpdf(grid_file_name)
q=100.
e_com = 7e9

sigma = 0.
dcosTheta = 0.05
cosThetas = np.arange(-1,1,dcosTheta)

ds_dO = np.zeros((3,cosThetas.size))

for flavor in [1,2,3]:
    for icosTheta,cosTheta in enumerate(cosThetas):
        ds_dO_obj = qft.dSigmaDOmega_PP( e_com, pdf, q, flavor,
cosTheta  )

        integ = vegas.Integrator(regn)
        # adapt to the integrand; discard results
        integ(ds_dO_obj, nitn=5, neval=1000)

        # do the final integral
        result = integ(ds_dO_obj, nitn=5, neval=50000)

        ds_dO[flavor-1,icosTheta] = result.mean
        sigma += result.mean * dcosTheta;
        # Produce qqbar --> Z --> e+e-
        #print( "%2d : %6.2f %10.6e" % (flavor, cosTheta, result
.mean))
print ("Total cross section: ", sigma)
```

```
Total cross section:  2.785449490467623e-15
```

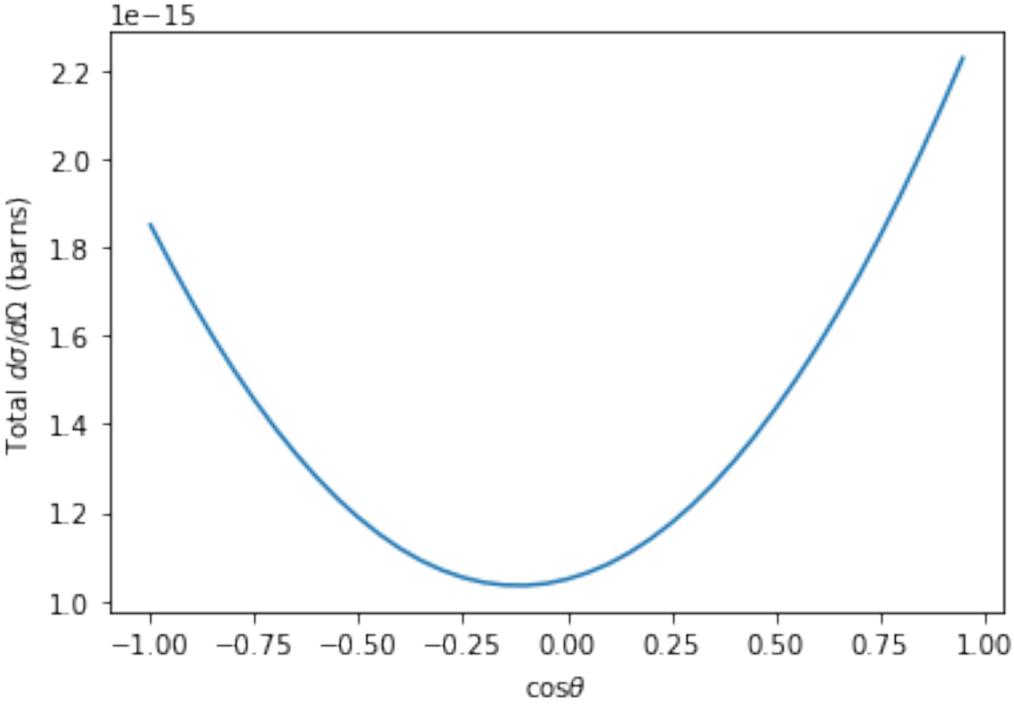
In [4]:

```
plt.plot(cosThetas, ds_dO.T )
plt.legend(['down','up','strange'])
plt.xlabel(r'$\cos{\theta}$')
plt.ylabel(r'Per-quark $d\sigma / d\Omega$ (barns)')
plt.show()
```



In [5]:

```
plt.plot(cosThetas, np.sum(ds_dO,axis=0) )
plt.xlabel(r'$\cos{\theta}$')
plt.ylabel(r'Total $d\sigma / d\Omega$ (barns)')
plt.show()
```



In [ ]: