# Basic Cross-Section Processing Using Python

In this exercise we will learn how to process data using ENDF data downloaded from the National Nuclear Data Center (nndc.bnl.gov). What I've done is download some data in comma-separated value tables and the first thing we do is read them in.

In this exercise we will consider a system made of pure $^{235}$U metal, similar to the Godiva reactor. Such a reactor will be a fast reactor, and we do not need to worry about the thermal part of the spectrum.

This first block of code uses a built-in function to read in the files. The second file has a header that we skip with the appropriate parameter.

```
In [1]: import numpy as np
        #open total cross-section
        sigma_t = np.genfromtxt('u235_total.csv', delimiter=",")
        #open scattering cross-section
        sigma_s = np.genfromtxt('u235_elastic.csv', delimiter=",", skip_header=1)
```

Different cross-section files have different points of evaluation (i.e., the energies are different). We can handle this by computing the union of the two grids.

```
In [2]: #get the union of the energy grids
        energies = np.union1d(sigma_t[:,0], sigma_s[:,0])
```

We will use an analytic expression for the fission spectrum $\chi(E)$.

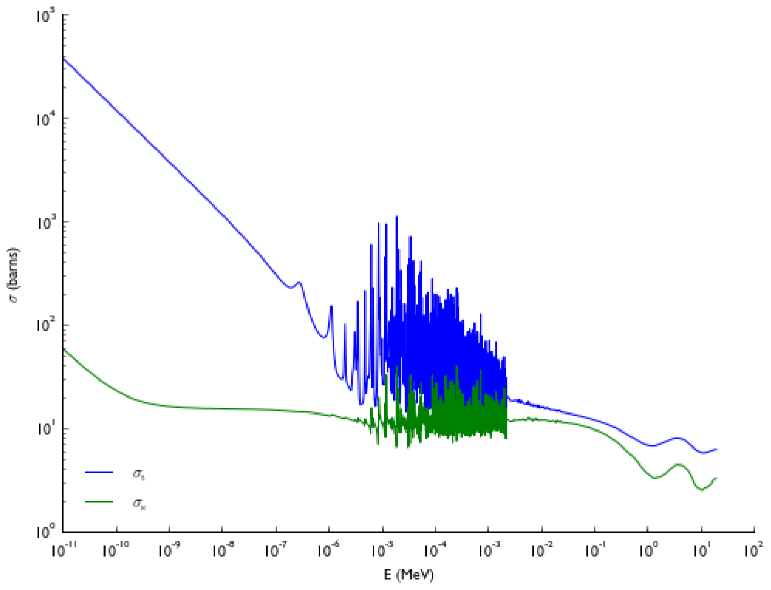
```
In [3]: #create the fission spectrum
        chi = lambda E:  0.4865*np.sinh(np.sqrt(2*E))*np.exp(-E)
```

Using SciPy we create functions to get the cross-sections at any energy via interpolation. We have to be care about evaluating the cross-sections above the maximum energy in the table. In our case we are likely to be evaluating above the max energy, so we set the cross-section to be the cross-section at the max energy.

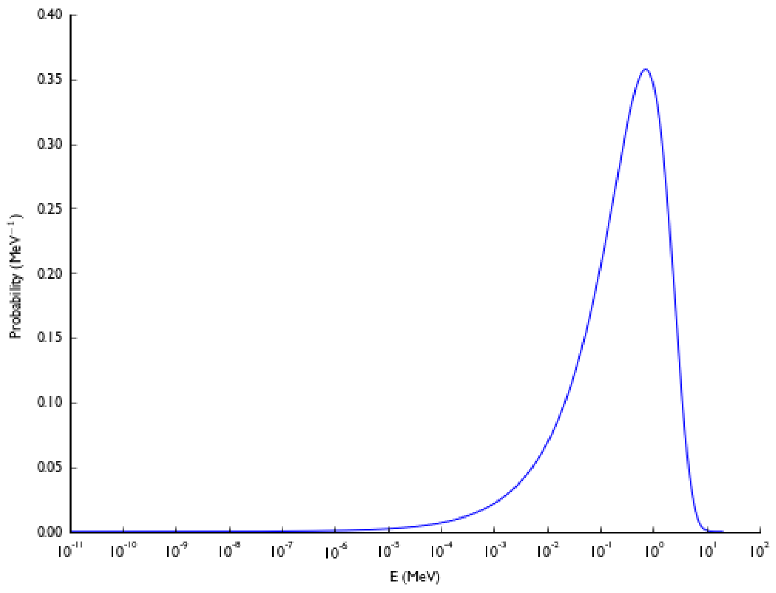
```
In [4]: #make interpolation functions
        from scipy import interpolate
        sig_t_interp = interpolate.interp1d(sigma_t[:,0], sigma_t[:,1],bounds_error=False, fill_value=sigm
        a_t[-1,1])
        sig_s_interp = interpolate.interp1d(sigma_s[:,0], sigma_s[:,1],bounds_error=False, fill_value=sigm
        a_s[-1,1])
```

As a sanity check let's plot the cross-sections and the fission spectrum.

In [5]:
```
#let's make some plots
%matplotlib inline

import matplotlib
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import math
import matplotlib.font_manager as fm
import matplotlib.ticker as mtick
font = fm.FontProperties(family = 'Gill Sans', fname = '/Library/Fonts/GillSans.ttc')
def hide_spines(intx=False,inty=False):
    """Hides the top and rightmost axis spines from view for all active
    figures and their respective axes."""

    # Retrieve a list of all current figures.
    figures = [x for x in matplotlib._pylab_helpers.Gcf.get_all_fig_managers()]
    if (plt.gca().get_legend()):
        plt.setp(plt.gca().get_legend().get_texts(), fontproperties=font)
    for figure in figures:
        # Get all Axis instances related to the figure.
        for ax in figure.canvas.figure.get_axes():
            # Disable spines.
            ax.spines['right'].set_color('none')
            ax.spines['top'].set_color('none')
            # Disable ticks.
            ax.xaxis.set_ticks_position('bottom')
            ax.yaxis.set_ticks_position('left')
           # ax.xaxis.set_major_formatter(mtick.FuncFormatter(lambda v,_: ("10$^{%d}$" % math.log(v,10))
))
            for label in ax.get_xticklabels() :
                label.set_fontproperties(font)
            for label in ax.get_yticklabels() :
                label.set_fontproperties(font)
            #ax.set_xticklabels(ax.get_xticks(), fontproperties = font)
            ax.set_xlabel(ax.get_xlabel(), fontproperties = font)
            ax.set_ylabel(ax.get_ylabel(), fontproperties = font)
            ax.set_title(ax.get_title(), fontproperties = font)
            if (inty):
                ax.yaxis.set_major_formatter(mtick.FormatStrFormatter('%d'))
            if (intx):
                ax.xaxis.set_major_formatter(mtick.FormatStrFormatter('%d'))
def show(nm=0,a=0,b=0):
    hide_spines(a,b)
    #ax.xaxis.set_major_formatter(mtick.FuncFormatter(lambda v,_: ("10$^{%d}$" % math.log(v,10)) ))
    #plt.yticks([1,1e-2,1e-4,1e-6,1e-8,1e-10,1e-12], labels)
    #ax.yaxis.set_major_formatter(mtick.FuncFormatter(lambda v,_: ("10$^{%d}$" % math.log(v,10)) ))
    if (nm != 0):
        plt.savefig(nm);
    plt.show()

fig = plt.figure(figsize=(8,6), dpi=1600)
plt.loglog(energies, sig_t_interp(energies), label="$\sigma_\mathrm{t}$")
plt.loglog(energies, sig_s_interp(energies), label="$\sigma_\mathrm{s}$")
plt.legend(loc=3) #bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.ylabel("$\sigma$ (barns)")
plt.xlabel("E (MeV)")
show("U-235_xsect.pdf")
```



```
In [6]: fig = plt.figure(figsize=(8,6), dpi=1600)
        plt.semilogx(energies,chi(energies))
        plt.xlabel("E (MeV)")
        plt.ylabel("Probability (MeV$^{-1}$)")
        show("U-235_chi.pdf")
```



## Computing the Spectrum for this reactor

We will attempt to compute the spectrum for this reactor using only the fine group cross-sections and the fission spectrum. To do this we look at the steady-state infinite medium problem given by

$$N^{235}\sigma_{\mathrm{t}}(E)\psi(\mu,E) = \frac{1}{2}\int_0^\infty dE'\, N^{235}\sigma_{\mathrm{s}}(E'\to E)\phi(E') + \frac{\chi(E)}{2k}\int_0^\infty dE'\, N^{235}\bar{\nu}\sigma_{\mathrm{f}}(E')\phi(E')$$

Note that in this equation we can normalize the fission term so that the effective source has a magnitude of $\chi(E)$:

$$\sigma_{\mathrm{t}}\psi(\mu,E) = \frac{1}{2}\int_0^\infty \sigma_{\mathrm{s}}(E'\to E)\phi(E') + \frac{\chi(E)}{2}.$$

For the energy transfer via scattering, we assume that all collisions have the same fractional energy loss:

$$\left(\frac{E}{E'}\right)_{\text{average}} = \frac{A^2+1}{(A+1)^2} \approx 0.991561333.$$

This allows us to write the transport equation, integrated over energy to be

$$\sigma_t(E)\phi(E) = \sigma_s\left(\frac{(A+1)^2}{A^2+1}E\right)\phi\left(\frac{(A+1)^2}{A^2+1}E\right) + \chi(E).$$

We can solve this equation at all the points in our energy grid using the following iteration procedure

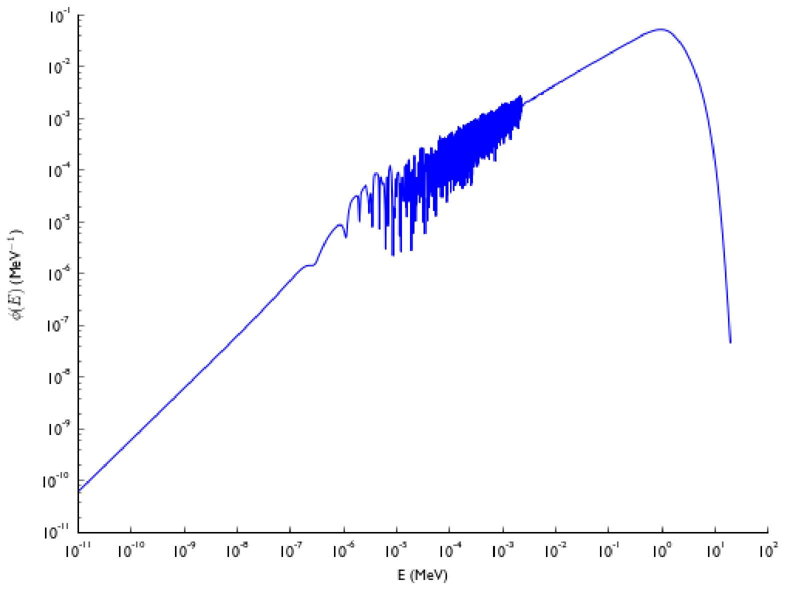
$$\phi^{l+1}(E) = \frac{1}{\sigma_t(E)}\sigma_s\left(\frac{(A+1)^2}{A^2+1}E\right)\phi^l\left(\frac{(A+1)^2}{A^2+1}E\right) + \frac{\chi(E)}{\sigma_t(E)}.$$

This is equivalent to a Jacobi iteration. Do we know that the operator is diagonally dominant?

For the first iteration we will assume that $\phi^0(E)$ is zero, so that

$$\phi^1(E) = \frac{\chi(E)}{\sigma_t(E)}.$$

In [7]:
```
#Now do 1 iteration
fig = plt.figure(figsize=(8,6), dpi=1600)
phi = interpolate.interp1d(energies,chi(energies)/sig_t_interp(energies),fill_value=1e-10,bounds_error=Fal
se)
plt.loglog(energies,phi(energies))
plt.xlabel("E (MeV)")
plt.ylabel("$\phi(E)$ (MeV$^{-1}$)")
show("FirstIteration.pdf")
```

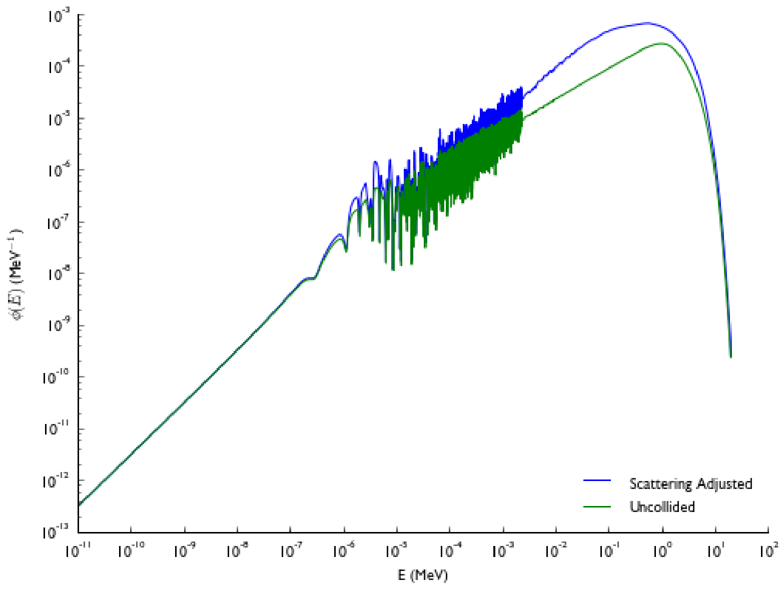


The next step is to do the iterations until we converge.

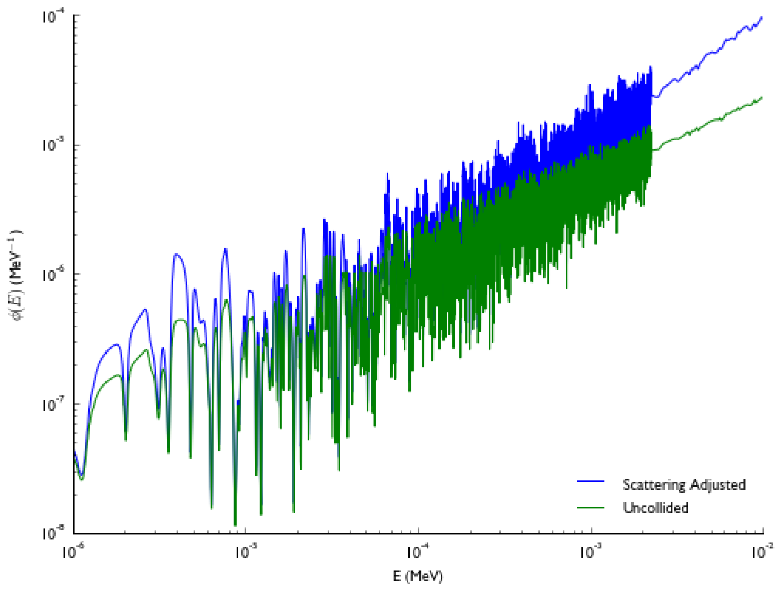
In [8]:
```
#converge the spectrum
A = 235
phi_iteration = lambda E: phi(E)
converged = 0
tolerance = 1.0e-6
iteration = 0
change_factor = (A+1)**2/(A**2+1)
while not(converged):
    phi_prev = interpolate.interp1d(energies,phi_iteration(energies),fill_value=0,bounds_error=False)
    phi_iteration= lambda E: (phi_prev(E*change_factor)*sig_s_interp(E*change_factor) + chi(E))/sig_t_inte
rp(E)
    converged = (np.linalg.norm(phi_prev(energies) - phi_iteration(energies))/
                np.linalg.norm(phi_iteration(energies)) < tolerance)
    iteration += 1
print("Number of iterations",iteration)
```

```
Number of iterations 55
```

In [9]:
```
fig = plt.figure(figsize=(8,6), dpi=1600)
plt.loglog(energies,phi_iteration(energies)/np.sum(phi_iteration(energies)), label="Scattering Adjusted")
plt.loglog(energies,phi(energies)/np.sum(phi_iteration(energies)), label="Uncollided")
plt.xlabel("E (MeV)")
plt.ylabel("$\phi(E)$ (MeV$^{-1}$)")
plt.legend(loc=4)
show("SpectrumComparison.pdf")
```



In [10]:
```
fig = plt.figure(figsize=(8,6), dpi=1600)
plt.loglog(energies,phi_iteration(energies)/np.sum(phi_iteration(energies)), label="Scattering Adjusted")
plt.loglog(energies,phi(energies)/np.sum(phi_iteration(energies)), label="Uncollided")
plt.xlabel("E (MeV)")
plt.ylabel("$\phi(E)$ (MeV$^{-1}$)")
plt.legend(loc=4)
plt.xlim([1e-6,1e-2])
plt.ylim([1e-8,1e-4])
show("SpectrumComparison_zoom.pdf")
```



## Uranium Oxide Instead of Uranium Metal

In this case the solution will be a little different because the oxygen in the fuel will do some moderating of the spectrum. For $U0_2$, the molar mass is 270.03 g/mol. We will assume 10%-atom enrichment of the fuel. This means that for every 1 nucleus of $^{238}$U, we have 0.1 nuclei of $^{235}$U, and 2 nuclei of $^{16}$O.

This makes our transport equation

$$\left[N^{238}\sigma_\mathrm{t}^{238}(E) + 0.1N^{238}\sigma_\mathrm{t}^{235}(E) + 2N^{238}\sigma_\mathrm{t}^{16}(E)\right]\psi(\mu,E) = \frac{1}{2}\int_0^\infty dE'\ \left[N^{238}\sigma_\mathrm{s}^{238}(E'\rightarrow E) + 0.1N^{238}\sigma_\mathrm{s}^{235}(E'\rightarrow E) + 2N^{238}\sigma_\mathrm{s}^{16}(E'\rightarrow E)\right]\phi(E') + \frac{\chi(E)}{2k}\int_0^\infty dE'\ \left[N^{238}\bar{\nu}\sigma_\mathrm{f}^{238}(E') + 0.1N^{238}\bar{\nu}\sigma_\mathrm{f}^{235}(E)\right]\phi(E')$$

We can make the same normalization to the fission source as we did above, and we will assume that the scattering all goes to the average energy to get

$$\left[\sigma_\mathrm{t}^{238}(E) + 0.1\sigma_\mathrm{t}^{235}(E) + 2\sigma_\mathrm{t}^{16}(E)\right]\phi(E) = \sigma_\mathrm{s}^{238}\left(\frac{(239)^2}{238^2+1}E\right)\phi\left(\frac{(239)^2}{238^2+1}E\right) + 0.1\sigma_\mathrm{s}^{235}\left(\frac{(236)^2}{235^2+1}E\right)\phi\left(\frac{(236)^2}{235^2+1}E\right) + 0.1\sigma_\mathrm{s}^{16}\left(\frac{(17)^2}{16^2+1}E\right)\phi\left(\frac{(17)^2}{16^2+1}E\right) + \chi(E)$$

We have also assumed that $\chi(E)$ has not changed.

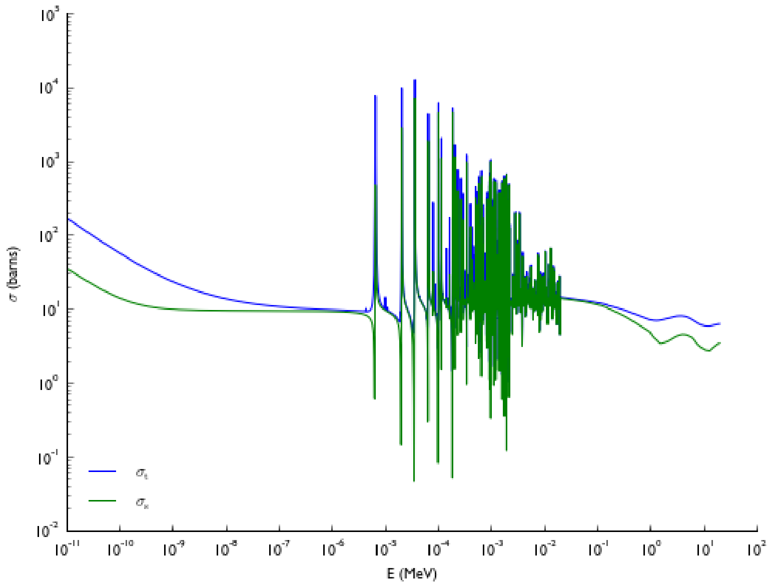
The next code snippet handles some reading in of data.

```
In [11]: #give previous functions useful names
         sig_s_235_interp = sig_s_interp
         sig_t_235_interp = sig_t_interp

         #read in 238-U data
         #open total cross-section
         sigma_t_238 = np.genfromtxt('u238_total.csv', delimiter=",")
         #open total cross-section
         sigma_s_238 = np.genfromtxt('u238_elastic.csv', delimiter=",")

         #read in 16-O data
         sigma_t_16 = np.genfromtxt('o16_total.csv', delimiter=",")
         #open total cross-section
         sigma_s_16 = np.genfromtxt('o16_elastic.csv', delimiter=",")

         #create interpolation functions
         sig_t_238_interp = interpolate.interp1d(sigma_t_238[:,0], sigma_t_238[:,1],bounds_error=False, fill_valu
         e=sigma_t_238[-1,1])
         sig_s_238_interp = interpolate.interp1d(sigma_s_238[:,0], sigma_s_238[:,1],bounds_error=False, fill_valu
         e=sigma_s_238[-1,1])
         sig_t_16_interp = interpolate.interp1d(sigma_t_16[:,0], sigma_t_16[:,1],bounds_error=False, fill_value=sig
         ma_t_16[-1,1])
         sig_s_16_interp = interpolate.interp1d(sigma_s_16[:,0], sigma_s_16[:,1],bounds_error=False, fill_value=sig
         ma_s_16[-1,1])

         energies_new = np.union1d(energies, sigma_t_238[:,0])
         energies = energies.copy()
```

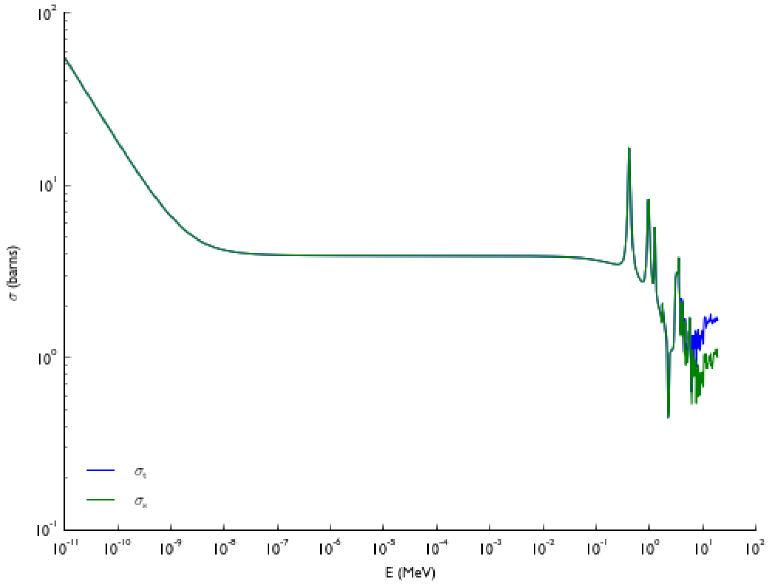
We'll plot the cross-sections as a sanity check.

In [12]:
```
fig = plt.figure(figsize=(8,6), dpi=1600)
plt.loglog(energies, sig_t_238_interp(energies), label="$\sigma_\mathrm{t}$")
plt.loglog(energies, sig_s_238_interp(energies), label="$\sigma_\mathrm{s}$")
plt.legend(loc=3) #bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.ylabel("$\sigma$ (barns)")
plt.xlabel("E (MeV)")
show("U-238_xsect.pdf")
```

```
/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages/scipy/interpolate/interpolat
e.py:520: RuntimeWarning: invalid value encountered in greater
  above_bounds = x_new > self.x[-1]
```



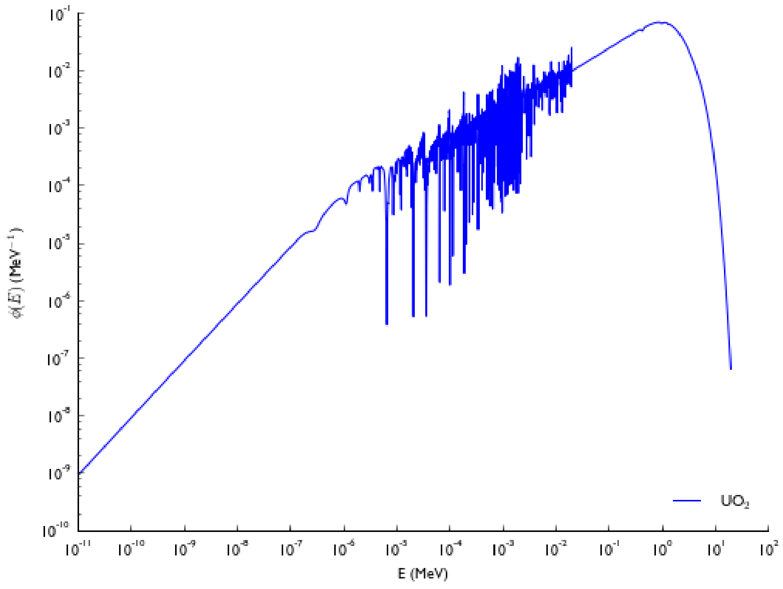
In [13]:
```
fig = plt.figure(figsize=(8,6), dpi=1600)
plt.loglog(energies, sig_t_16_interp(energies), label="$\sigma_\mathrm{t}$")
plt.loglog(energies, sig_s_16_interp(energies), label="$\sigma_\mathrm{s}$")
plt.legend(loc=3) #bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.ylabel("$\sigma$ (barns)")
plt.xlabel("E (MeV)")
show("O-16_xsect.pdf")
```



We will do the same iteration strategy as before. Now the initial spectrum is given by

$$\left[\sigma_\mathrm{t}^{238}(E) + 0.1\sigma_\mathrm{t}^{235}(E) + 2\sigma_\mathrm{t}^{16}(E)\right]\phi(E) = \chi(E).$$

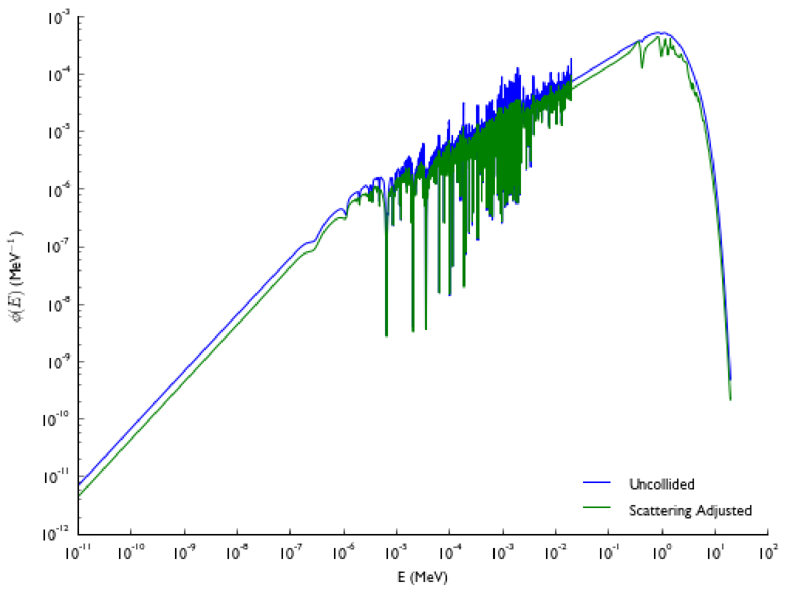
```
In [14]: #Now do 1 iteration
         fig = plt.figure(figsize=(8,6), dpi=1600)
         phi_oxide = interpolate.interp1d(energies,chi(energies)/
                                   (sig_t_238_interp(energies) + 0.1 * sig_t_235_interp(energies) + 0.1 * sig_t_1
         6_interp(energies))
                                    ,fill_value=1.0e-11,bounds_error=False)
         plt.loglog(energies,phi_oxide(energies)/np.linalg.norm(phi_oxide(energies)),label="UO$_2$")
         #plt.loglog(energies,phi(energies)/np.linalg.norm(phi(energies)),label="U metal")
         plt.xlabel("E (MeV)")
         plt.ylabel("$\phi(E)$ (MeV$^{-1}$)")
         plt.legend(loc=4)
         show("FirstIteration_oxide.pdf")
```



In [15]:
```
#converge the spectrum
phi_iterationox = lambda E: phi_oxide(E)
converged = 0
tolerance = 1.0e-6
iteration = 0
max_iterations = 100
change_factor_func = lambda A: (A+1)**2/(A**2+1)
#first solve without 238-U scattering
while not(converged):
    phi_prev = interpolate.interp1d(energies,phi_iterationox(energies),fill_value=0,bounds_error=False)
    phi_iterationox= lambda E: (0*phi_prev(E*change_factor_func(238))*sig_s_238_interp(E*change_factor_fun
c(238))
                                + 0.1 * phi_prev(E*change_factor_func(235))*sig_s_235_interp(E*change_fact
or_func(235))
                                + 2 * phi_prev(E*change_factor_func(16))*sig_s_16_interp(E*change_factor_f
unc(16))
                                + chi(E))/(sig_t_238_interp(energies) +
                                           0.1 * sig_t_235_interp(energies) +
                                           2 * sig_t_16_interp(energies))
    converged = (np.linalg.norm(phi_prev(energies) - phi_iterationox(energies))/
                 np.linalg.norm(phi_iterationox(energies)) < tolerance) or (iteration >= max_iterations)
    iteration += 1
print("Number of iterations",iteration)

#using that as initial guess, now solve the entire thing
while not(converged):
    phi_prev = interpolate.interp1d(energies,phi_iterationox(energies),fill_value=0,bounds_error=False)
    phi_iterationox= lambda E: (phi_prev(E*change_factor_func(238))*sig_s_238_interp(E*change_factor_fun
c(238))
                                + 0.1 * phi_prev(E*change_factor_func(235))*sig_s_235_interp(E*change_fact
or_func(235))
                                + 2 * phi_prev(E*change_factor_func(16))*sig_s_16_interp(E*change_factor_f
unc(16))
                                + chi(E))/(sig_t_238_interp(energies) +
                                           0.1 * sig_t_235_interp(energies) +
                                           2 * sig_t_16_interp(energies))
    converged = (np.linalg.norm(phi_prev(energies) - phi_iterationox(energies))/
                 np.linalg.norm(phi_iterationox(energies)) < tolerance) or (iteration >= max_iterations)
    iteration += 1
print("Number of iterations",iteration)
```

```
Number of iterations 16
Number of iterations 16
```

In [16]:
```
fig = plt.figure(figsize=(8,6), dpi=1600)
plt.loglog(energies,phi_oxide(energies)/np.sum(phi(energies)), label="Uncollided")
plt.loglog(energies,phi_iterationox(energies)/np.sum(phi_iterationox(energies)), label="Scattering Adjuste
d")
plt.xlabel("E (MeV)")
plt.ylabel("$\phi(E)$ (MeV$^{-1}$)")
plt.legend(loc=4)
show("SpectrumComparison_oxide.pdf")
```



## What does the spectrum look like for Uranium Hydride?

```
In [17]:
```