healpy tutorial

`healpy` is a python wrapper for a C++ library called HEALPix (Hierarchical Equal Area isoLatitude Pixelization). It was initially developed with CMB data analysis in mind, but is useful for any situation where you have to handle functions on the sphere (i.e. pretty much all of astronomy and observational cosmology).

Some useful links:

- healpy documentation (very thorough, should be your go-to when you get stuck) https://healpy.readthedocs.io/en/latest/ (https://healpy.readthedocs.io/en/latest/)
- original HEALPix documentation https://healpix.jpl.nasa.gov/index.shtml (https://healpix.jpl.nasa.gov/index.shtml)
- the HEALPix paper (with 2000+ citations!) https://inspirehep.net/record/659804 (https://inspirehep.net/record/659804)

Basics

Let's start by importing the `healpy` library, along with some other useful things.

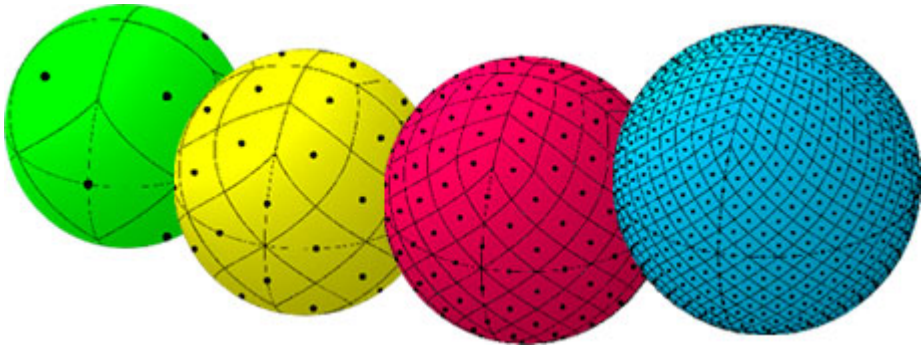
In [1]:
```
%matplotlib inline
import healpy as hp
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
plt.rcParams.update({"text.usetex":True})
plt.rcParams.update({"font.size":20})
```

`healpy` is, at its heart, an efficient algorithm for tiling the sphere into equal-area "pixels". We control the number of pixels (equivalently, the angular resolution of the map) using the "nside" parameter, which must be an integer power of 2, less than 2^{30}. Shown below are some HEALPix pixelisations for various values of nside.

In [3]:
```
from IPython.display import Image
from IPython.core.display import HTML
Image(url= "https://healpix.jpl.nasa.gov/images/healpixGridRefinement.jpg")
```

Out[3]:



Let's set a reasonable value of nside.

In [4]:
```
nside = 2**9
```

`healpy` provides some useful functions for converting this into the corresponding number of pixels, pixel area, and angular resolution.

In [5]:
```
npix = hp.nside2npix(nside)
pixarea = hp.nside2pixarea(nside)
resol = hp.nside2resol(nside)

print("the number of pixels is {}".format( npix ))
print("the pixel area in steradians is {:.2e}".format( pixarea ))
print("the angular resolution in radians is {:.2e}".format( resol ))
```

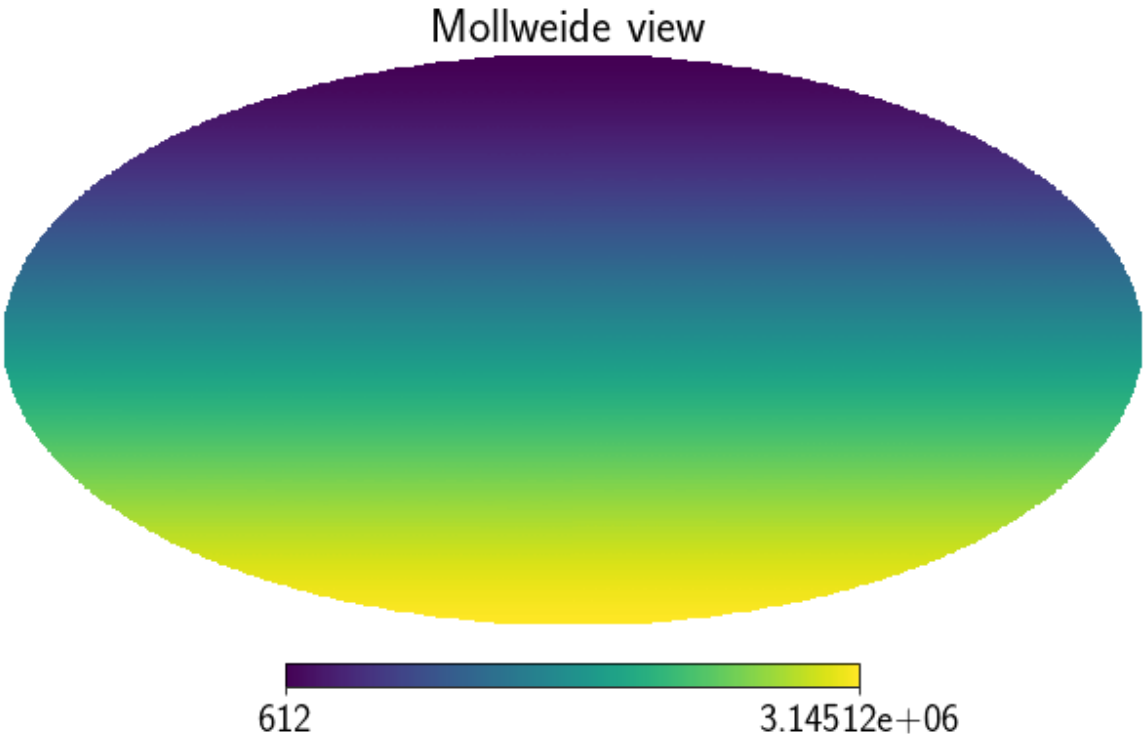
```
the number of pixels is 3145728
the pixel area in steradians is 3.99e-06
the angular resolution in radians is 2.00e-03
```

Let's create a map. A `healpy` map is just a `numpy` array, with each number in the array corresponding to the value in one pixel.

In [6]:
```
map_range = np.arange(npix)
```

In order to visualise this map, we use one of the plotting functions from the module `healpy.visufunc`. There are lots of different ways to project the sphere S^2 onto the plane $\mathbb{R}^2$ (like this screen), but let's use the Mollweide projection.

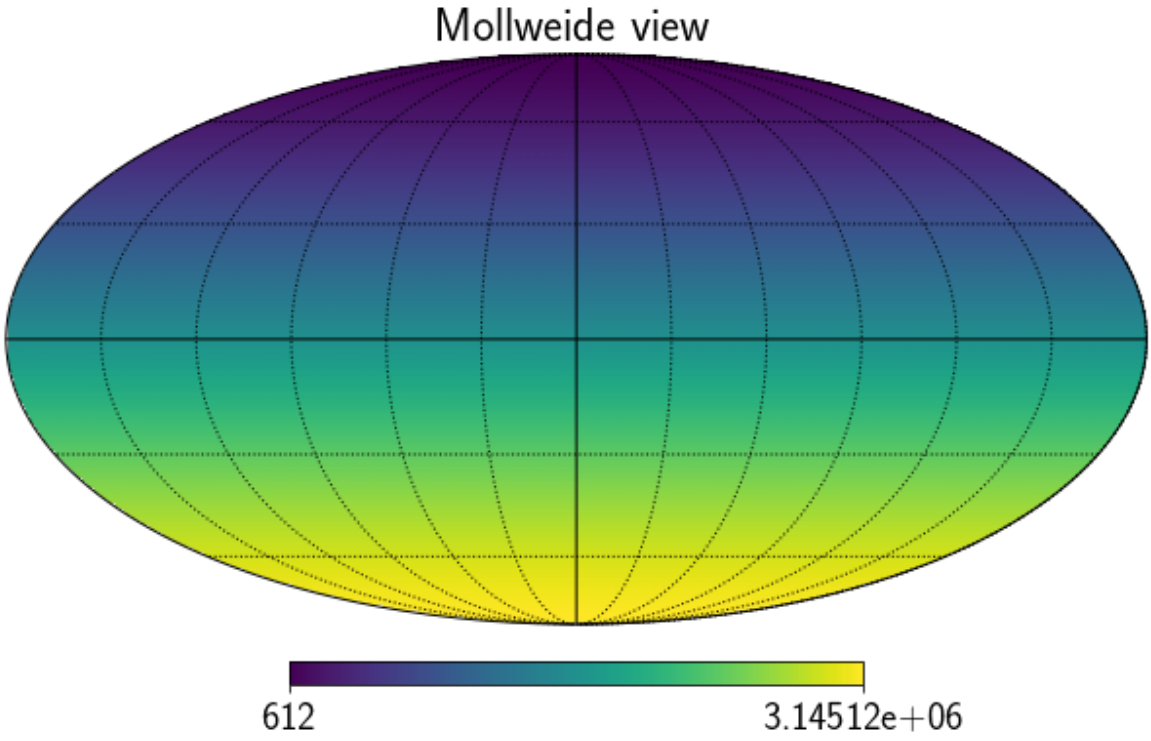
In [7]:
```
hp.mollview(map_range)
```



The `healpy.visufunc.graticule` function can be used to add grid lines.

In [8]:
```
hp.mollview(map_range)
hp.graticule()
```

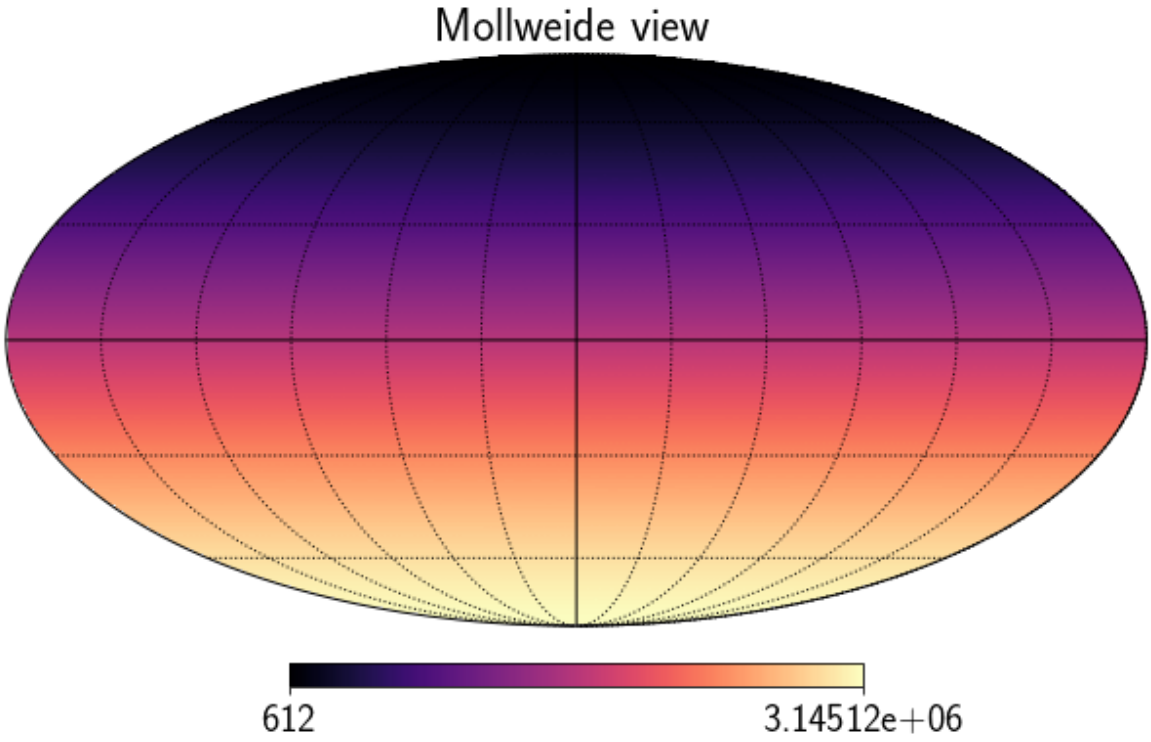
```
0.0 180.0 -180.0 180.0
The interval between parallels is 30 deg -0.00'.
The interval between meridians is 30 deg -0.00'.
```



We can also use the `cmap` argument to change the colourmap (the default is "viridis"; see https://matplotlib.org/tutorials/colors/colormaps.html (https://matplotlib.org/tutorials/colors/colormaps.html) for a full list).

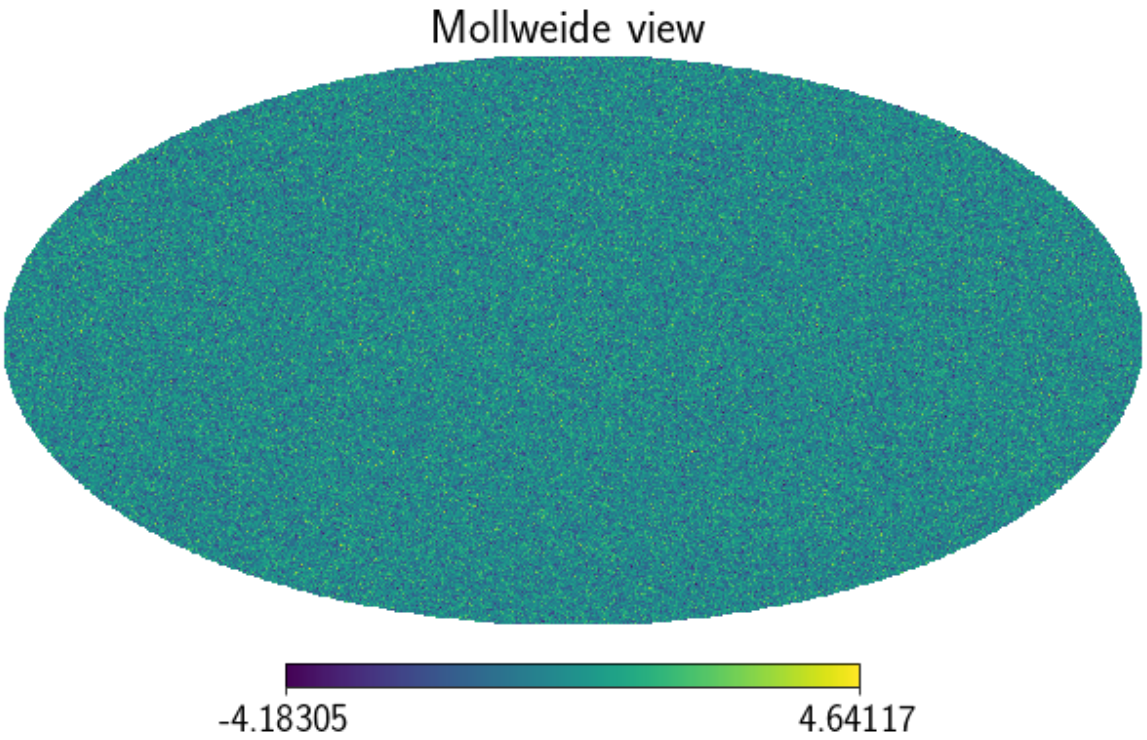
In [9]:
```
hp.mollview(map_range, cmap="magma")
hp.graticule()
```

```
0.0 180.0 -180.0 180.0
The interval between parallels is 30 deg -0.00'.
The interval between meridians is 30 deg -0.00'.
```



This map is pretty artificial-looking. Let's simulate a Gaussian random noise map using `numpy`'s random number generator.

In [10]:
```
map_gauss = np.random.normal(size=npix)
hp.mollview(map_gauss)
```



`healpy` provides some useful functions for converting between a given pixel's index within the map array, its angular coordinates, and the unit vector pointing in its direction. Let's find the index of the pixel with coordinates $\theta = \pi/4$, $\phi = \pi/2$, then convert this to a vector and check it has unit length.

In [11]:
```
pix = hp.ang2pix(nside, np.pi/4.0, np.pi/2.0)
print("the pixel index for coordinates theta=pi/4, phi=pi/2 is {}".format(p
ix))

vec = hp.pix2vec(nside, pix)
print("the unit vector pointing to this pixel is {}".format(vec))

vec = hp.ang2vec(np.pi/4.0, np.pi/2.0)
print("the unit vector point to those coordinates is {}".format(vec))
print("this vector has length {}".format(vec.dot(vec)))
```

```
the pixel index for coordinates theta=pi/4, phi=pi/2 is 460320
the unit vector pointing to this pixel is (-0.0011571238232320875, 0.7071813
576359993, 0.70703125)
the unit vector point to those coordinates is [4.32978028e-17 7.07106781e-01
7.07106781e-01]
this vector has length 1.0
```

Note that the outputs of `pix2vec` and `ang2vec` are slightly different---this is because `pix2vec` returns a vector pointing toward the centre of the pixel, which is slightly offset from our original coordinates.

Also, notice that functions which depend on the pixelisation scheme must take `nside` as an argument.

Exercise 1: Find the angle between the minimum and maximum values of the Gaussian map we've created.

In [12]:
```
vec_max = np.array(hp.pix2vec(nside, map_gauss.argmax()))
vec_min = np.array(hp.pix2vec(nside, map_gauss.argmin()))
ang_sep = np.arccos(vec_max.dot(vec_min))
print("the maximum and minimum points are separated by an angle of {0:.3} r
adians".format(ang_sep))
```

```
the maximum and minimum points are separated by an angle of 1.32 radians
```

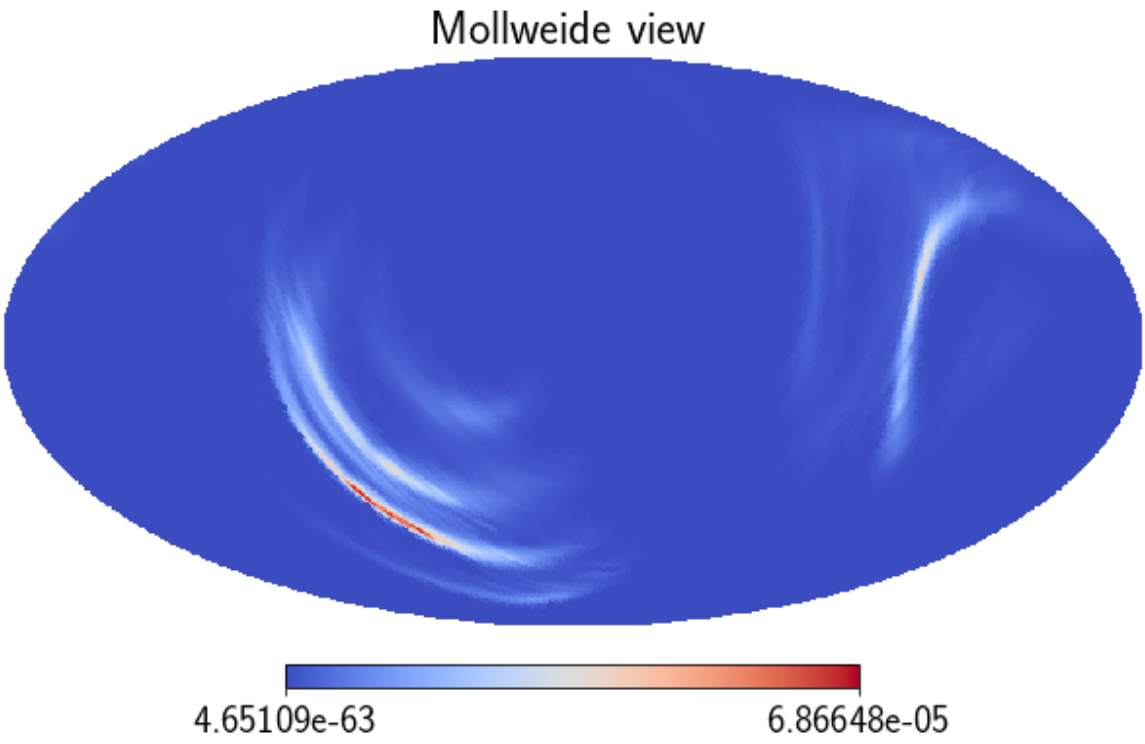
Fun with LIGO/Virgo alerts

Now for a fun example with some actual data. Let's download the publicly-available skymap for GW190425, a recent binary neutron star signal detected by LIGO/Virgo. This is just a HEALPix map of the probability distribution of the source's sky location.

The map is stored in a `.fits` file, which is a special file format often used in astronomy and cosmology for full-sky maps. We can read it into python using the `healpy.read_map` function, which will automatically determine the correct `nside` to use.

In [13]:
```
#! wget -v https://gracedb.ligo.org/api/superevents/S190425z/files/LALInfer
ence.fits.gz
map_gw = hp.read_map("LALInference.fits.gz")
nside = hp.get_nside(map_gw)
hp.mollview(map_gw, cmap="coolwarm")
```

```
NSIDE = 256
ORDERING = NESTED in fits file
INDXSCHM = IMPLICIT
Ordering converted to RING
```



We can check that the pixel values sum to unity, as they should for a probability distribution.

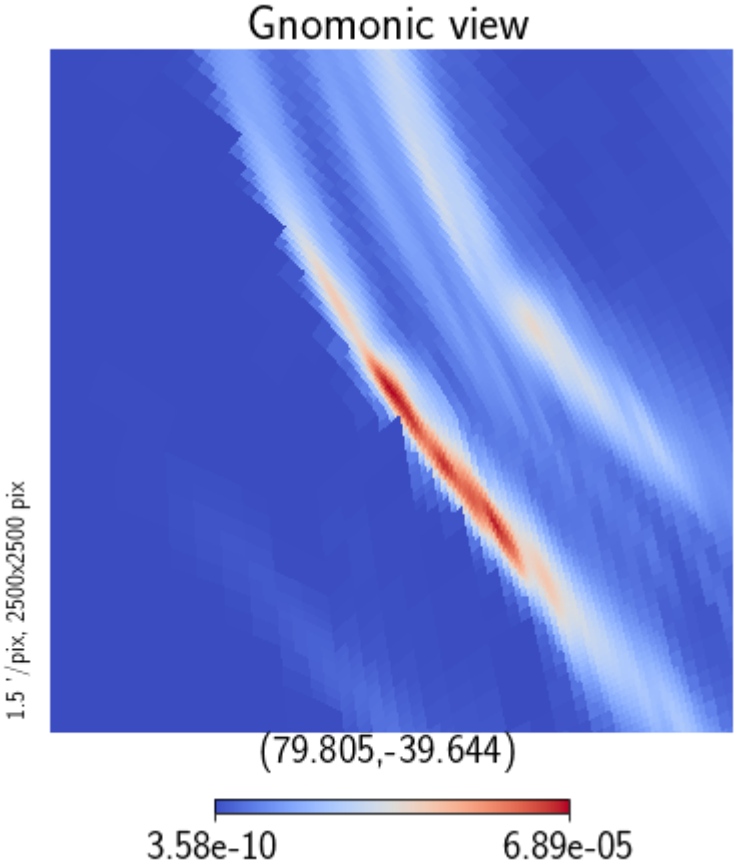
In [14]:
```
print np.sum(map_gw)
```

```
1.0000000000000002
```

Clearly the probability distribution is peaked in a relatively small area. We might want to zoom in on this area. To do so, we can use a gnomonic projection around a given point, using `healpy.gnomview`.

In the cell below we'll find the angular coordinates of the maximum-probability pixel, and zoom in to a 1000-by-1000 pixel area centred on that point. (Note the use of the "lonlat" argument, which ensures the angular coordinates are in the right format for `gnomview`.)

In [15]:
```
ang_max = hp.pix2ang(nside, map_gw.argmax(), lonlat=True)
hp.gnomview(map_gw, rot=ang_max, xsize=2500, ysize=2500, cmap="coolwarm")
```



Exercise 2: Estimate the size in steradians of the 90% confidence region---i.e. the smallest possible region of the sky which contains 90% of the total probability.

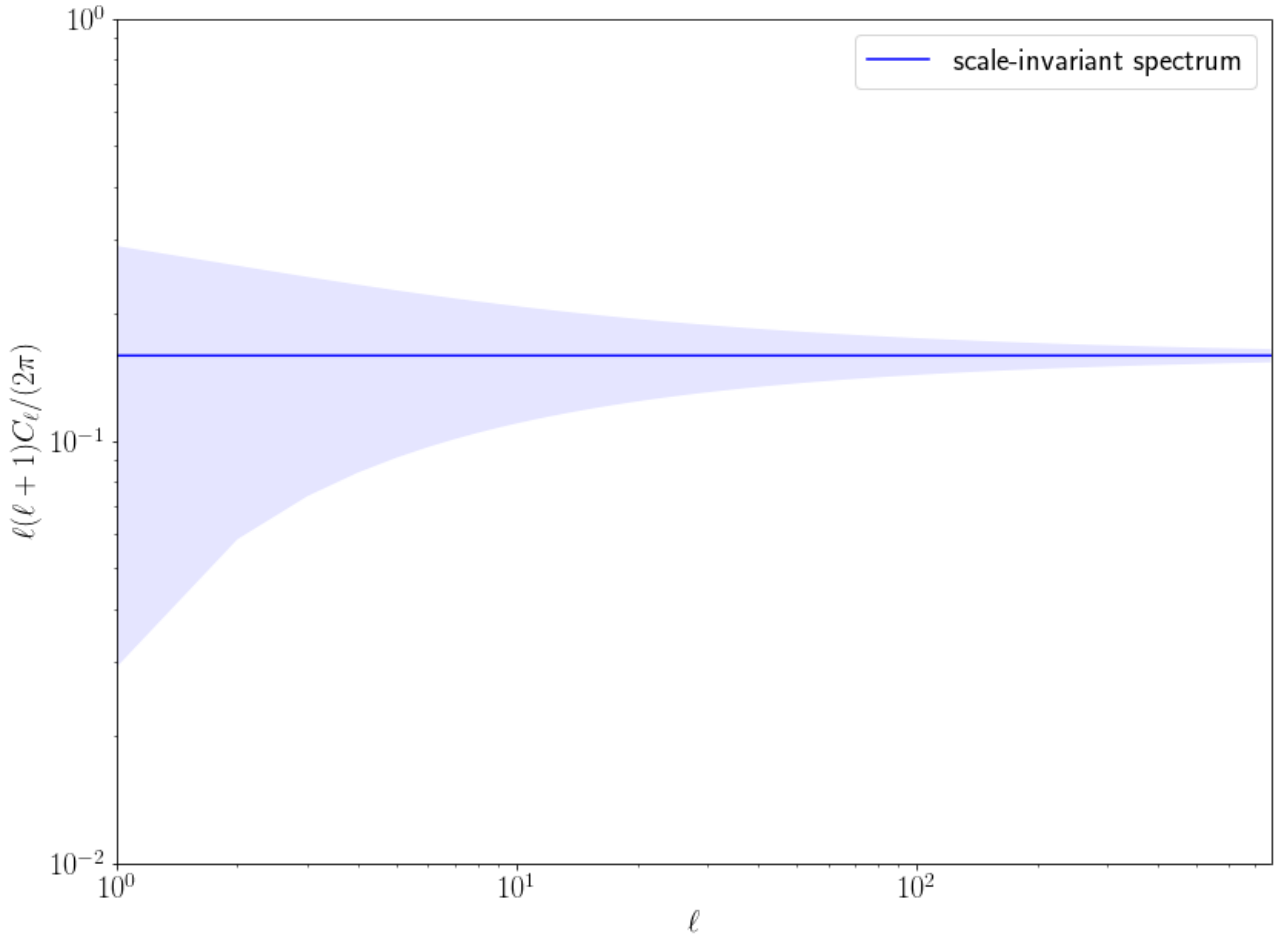
In [16]:
```
cum_prob = 0.0
pixel_count = 0
map_sort = list(np.sort(map_gw))

while cum_prob < 0.9:
    cum_prob += map_sort.pop()
    pixel_count += 1

pixarea = hp.nside2pixarea(nside)

print("the 90% confidence region is roughly {:.3} steradians".format(pixel_
count*pixarea))
```

```
the 90% confidence region is roughly 2.27 steradians
```

Spherical harmonics and angular power spectra

Let's start by defining a scale-invariant angular power spectrum. Note that the maximum ℓ we can use is set by the resolution of the maps we'll be working with, and is limited to $\ell_{\rm max} = 3 \times \mathtt{nside} - 1$.

In [17]:
```
lmax = 3*nside-1
ell = np.arange(lmax+1)
C_ell = np.array([np.power((l+1.0)*np.max([l,1.0]),-1.0) for l in ell])
sigma = np.array([np.sqrt(2.0*np.power(2.0*l+1.0,-1.0)) for l in ell])

D_ell = np.array([0.5*l*(l+1.0)*C_ell[l]/np.pi for l in ell])
D_ell_plus_sigma  = np.array([D_ell[l]*(1.0+sigma[l]) for l in ell])
D_ell_minus_sigma = np.array([D_ell[l]*(1.0-sigma[l]) for l in ell])

fig = plt.figure()
fig.set_size_inches(12, 9)
plt.xlabel(r"$\ell$")
plt.ylabel(r"$\ell(\ell+1)C_\ell/(2\pi)$")
plt.loglog(ell, D_ell, "b", label=r"scale-invariant spectrum")
plt.fill_between(ell, D_ell_plus_sigma,
                      D_ell_minus_sigma,
                      where=D_ell_plus_sigma>=D_ell_minus_sigma,
                      facecolor="b", alpha=0.1)
plt.xlim(1, lmax)
plt.ylim(0.01, 1.0)
plt.legend(loc=0)
fig.tight_layout()
```

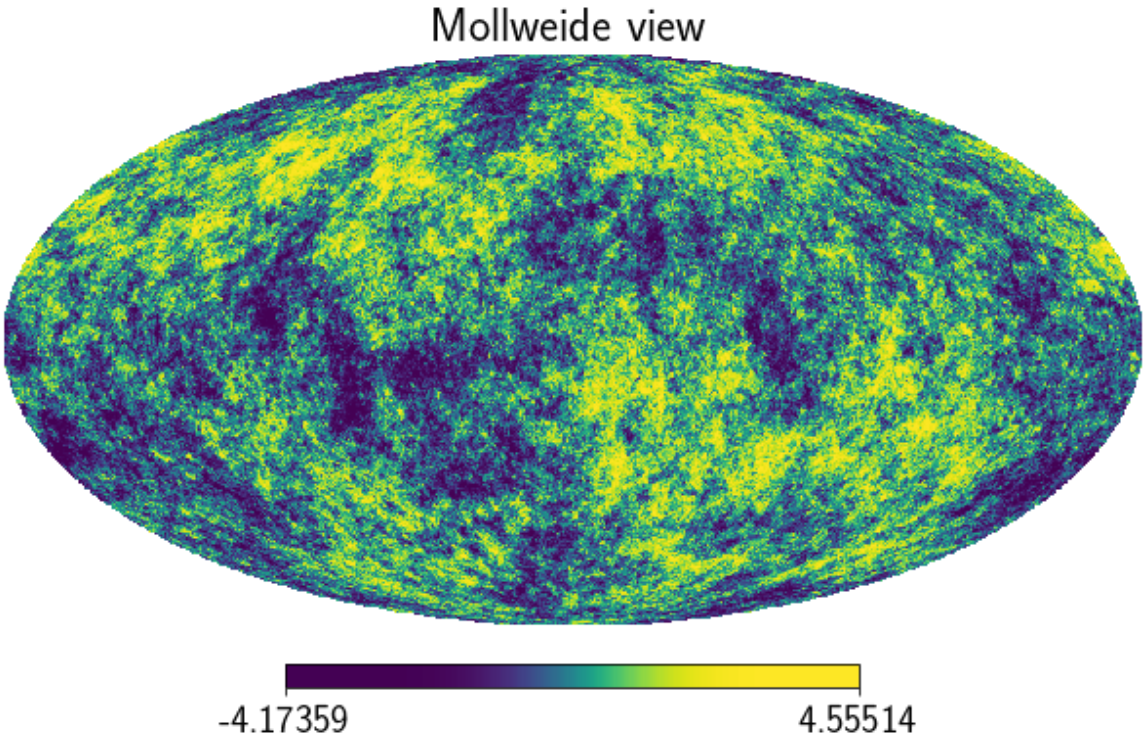


One of the most common things you'll want to do with `healpy` is to generate a map from a given spectrum. This is done using the function `healpy.synfast` (where the "syn" stands for "synthesise").

What we are doing here is effectively drawing each $a_{\ell m}$ from a Gaussian distribution with variance C_ℓ. Since this is a random process, notice that rerunning this cell leads to a different map each time.

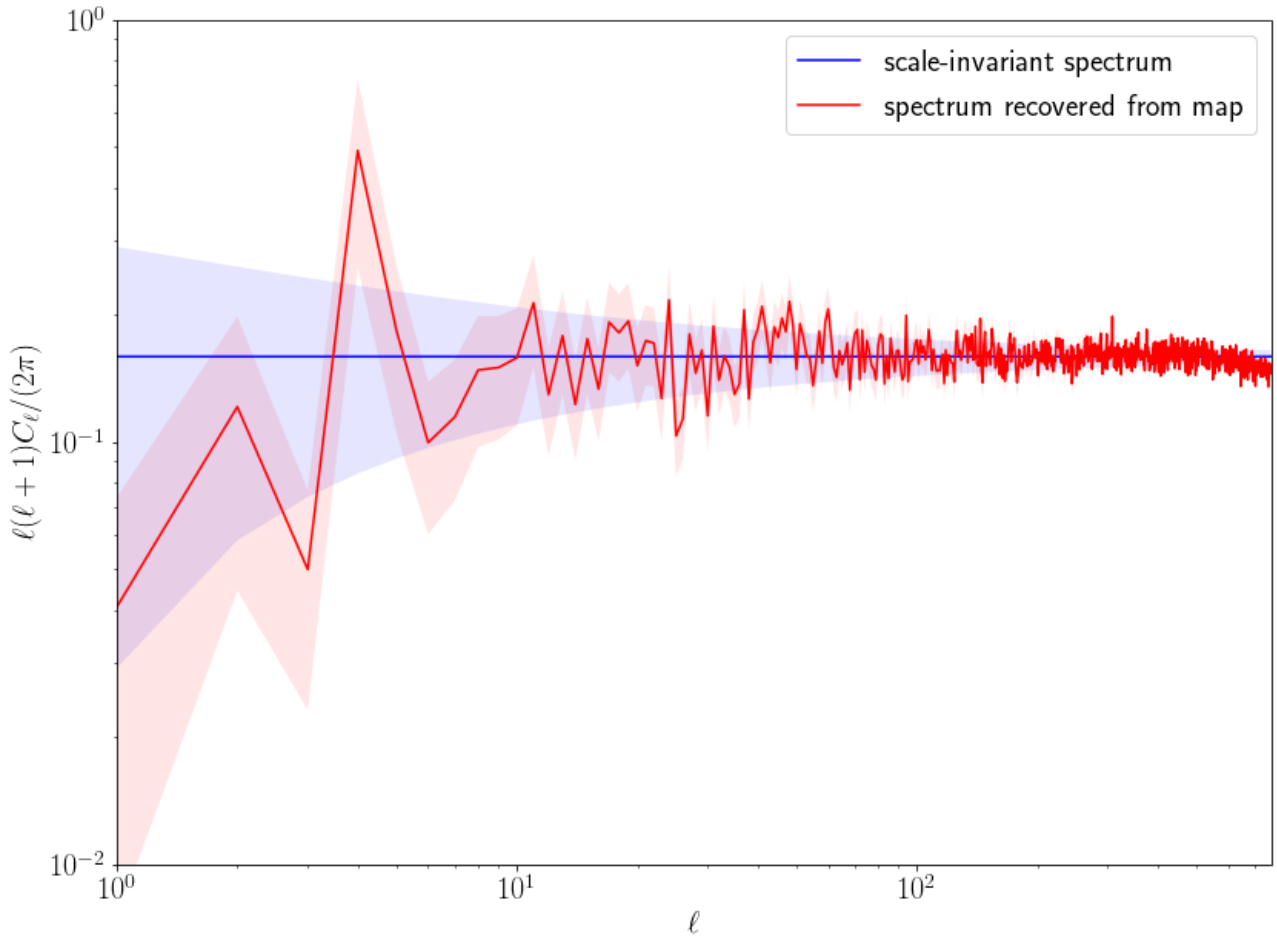
In [18]:
```
map_scale_inv = hp.synfast(C_ell, nside, new=True, verbose=False)

fig = plt.figure()
fig.set_size_inches(16, 9)
hp.mollview(map_scale_inv, norm="hist")
```

```
<Figure size 1152x648 with 0 Axes>
```



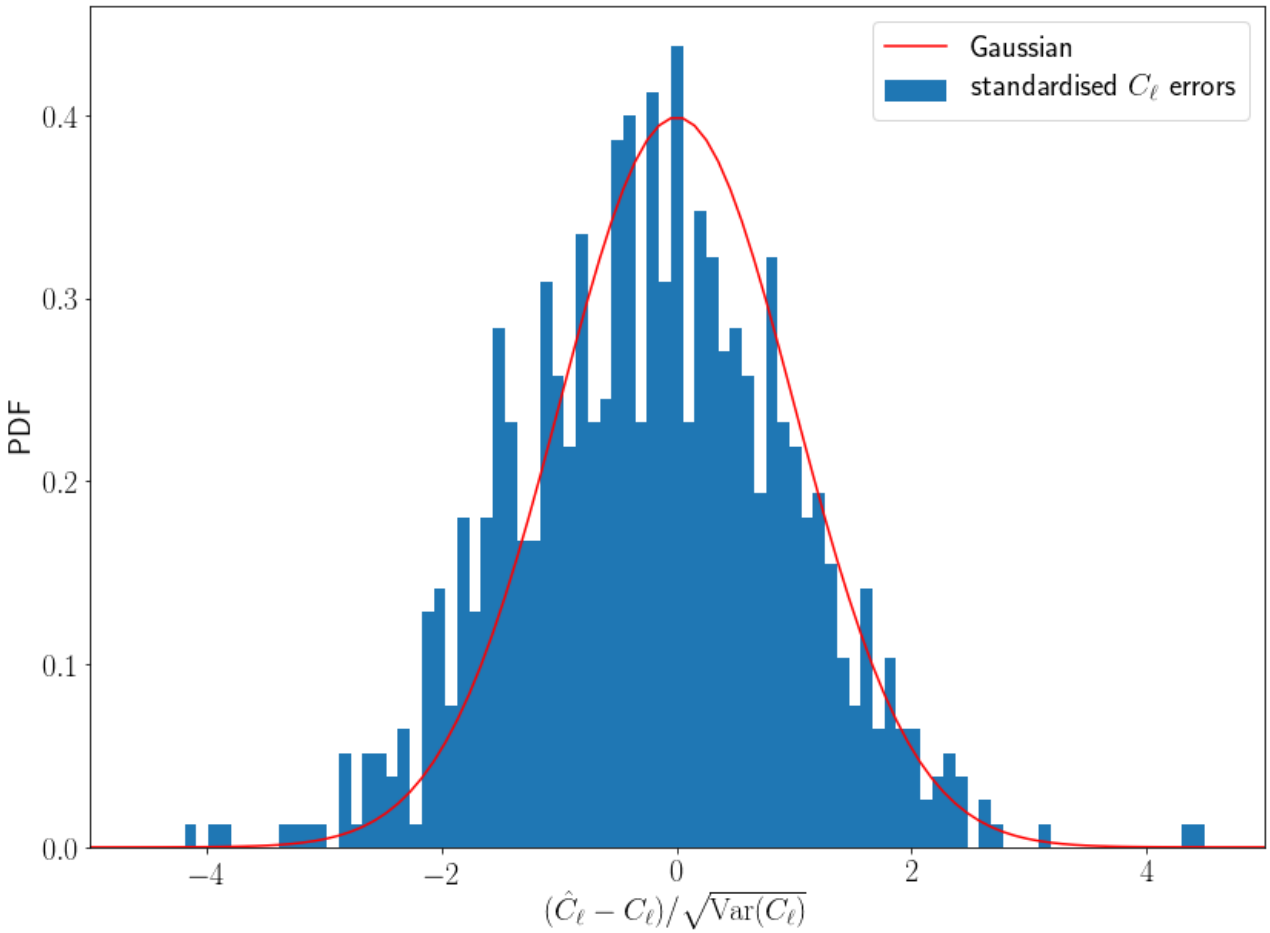
We'll also often want to do the reverse process: analyse a given map to find its spectrum. We do this with `healpy.anafast`.

In [19]:
```
C_ell_ana = hp.anafast(map_scale_inv)

D_ell_ana = np.array([0.5*l*(l+1.0)*C_ell_ana[l]/np.pi for l in ell])
D_ell_ana_plus_sigma  = np.array([D_ell_ana[l]*(1.0+sigma[l]) for l in ell
])
D_ell_ana_minus_sigma = np.array([D_ell_ana[l]*(1.0-sigma[l]) for l in ell
])

fig = plt.figure()
fig.set_size_inches(12, 9)
plt.xlabel(r"$\ell$")
plt.ylabel(r"$\ell(\ell+1)C_\ell/(2\pi)$")
plt.loglog(ell, D_ell, "b", label=r"scale-invariant spectrum")
plt.fill_between(ell, D_ell_plus_sigma,
                      D_ell_minus_sigma,
                      where=D_ell_plus_sigma>=D_ell_minus_sigma,
                      facecolor="b", alpha=0.1)
plt.loglog(ell, D_ell_ana, "r", label=r"spectrum recovered from map")
plt.fill_between(ell, D_ell_ana_plus_sigma,
                      D_ell_ana_minus_sigma,
                      where=D_ell_ana_plus_sigma>=D_ell_ana_minus_sigma,
                      facecolor="r", alpha=0.1)
plt.xlim(1, lmax)
plt.ylim(0.01, 1.0)
plt.legend(loc=0)
fig.tight_layout()
```



Notice that the recovered spectrum is *not* the same as the one we injected, due to cosmic variance. We should expect the points on the red line to be randomly distributed around the points on the blue line, with variance equal to $2C_\ell^2/(2\ell+1)$. Let's check this by creating a histogram.

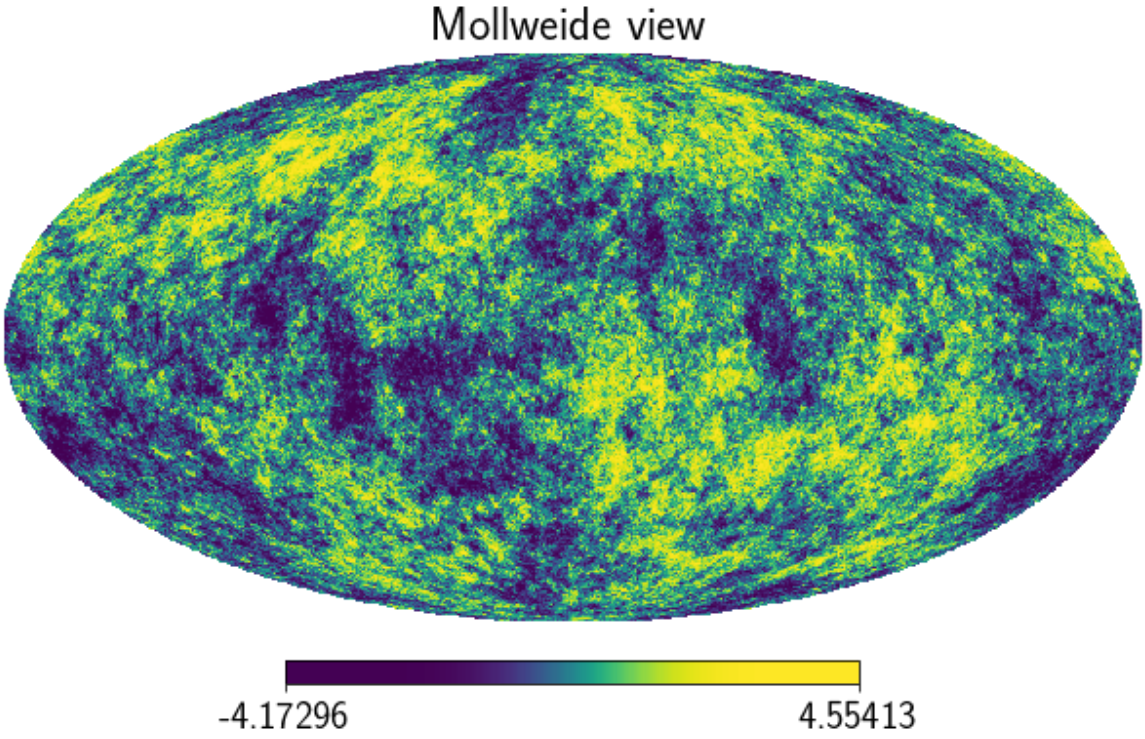
In [20]:
```
std_errors = np.array([(C_ell_ana[l]-C_ell[l])/(C_ell[l]*sigma[l]) for l in
ell])

bins = np.linspace(-5.0, +5.0, num=100)
gaussian = np.array([np.power(2.0*np.pi,-0.5)*np.exp(-0.5*bins[x]**2.0) for
x in xrange(100)])

fig = plt.figure()
fig.set_size_inches(12, 9)
plt.xlabel(r"$(\hat{C}_\ell-C_\ell)/\sqrt{\mathrm{Var}(C_\ell)}$")
plt.ylabel("PDF")
n, bins, patches = plt.hist(std_errors, bins=bins, density=True, label=r"st
andardised $C_\ell$ errors")
plt.plot(bins, gaussian, "r", label="Gaussian")
plt.xlim(-5.0, +5.0)
plt.legend(loc=0)
fig.tight_layout()
```

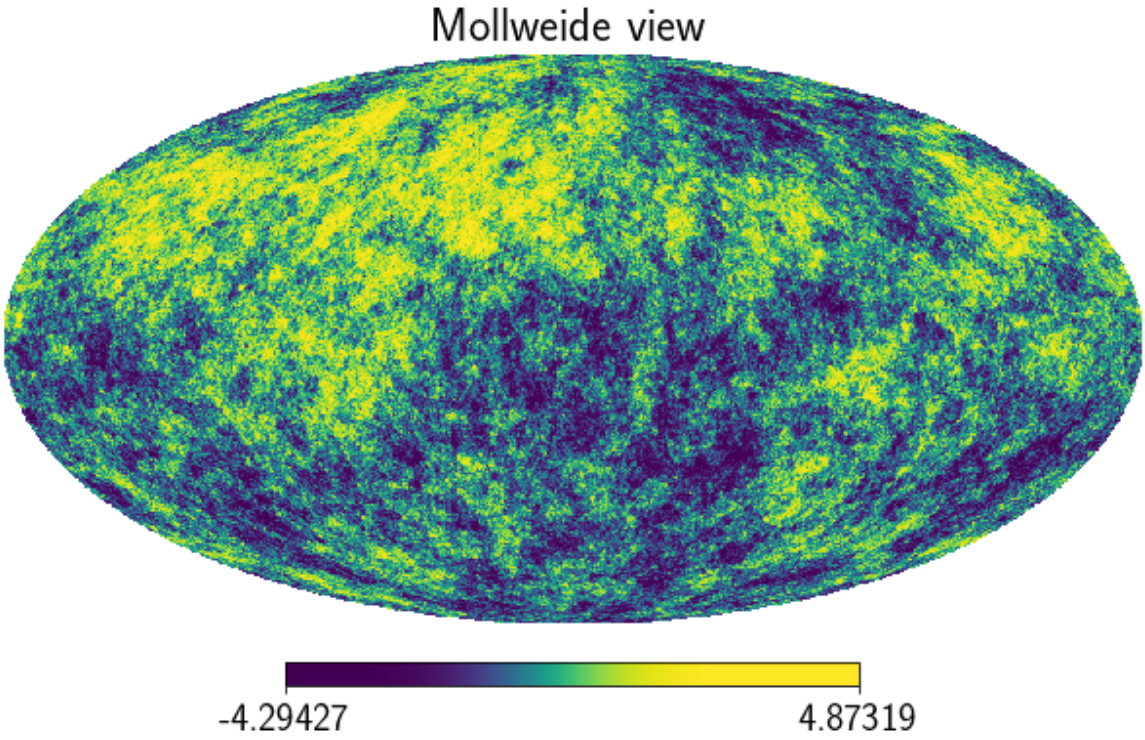


We may also want to work in terms of the $a_{\ell m}$'s. There are two ways to do this: either use **`healpy.map2alm`** to calculate the $a_{\ell m}$'s of a given map, or use **`healpy.synalm`** to randomly generate a set of $a_{\ell m}$'s from a given spectrum. The conversion from $a_{\ell m}$'s back to a map or a spectrum are achieved with **`healpy.alm2map`** and **`healpy.alm2cl`** respectively.

In [21]:
```
alm_map = hp.map2alm(map_scale_inv)
alm_syn = hp.synalm(C_ell, new=True)

hp.mollview(hp.alm2map(alm_map, nside), norm="hist")
hp.mollview(hp.alm2map(alm_syn, nside), norm="hist")
```

```
Sigma is 0.000000 arcmin (0.000000 rad)
-> fwhm is 0.000000 arcmin
Sigma is 0.000000 arcmin (0.000000 rad)
-> fwhm is 0.000000 arcmin
```





Both `alm_map` and `alm_syn` in the previous cell are `numpy` arrays, with each element corresponding to a particular (ℓ, m). However, the order that `healpy` puts these in within the array is pretty counter-intuitive, which can be annoying when you want to access a particular $a_{\ell m}$ from the array.

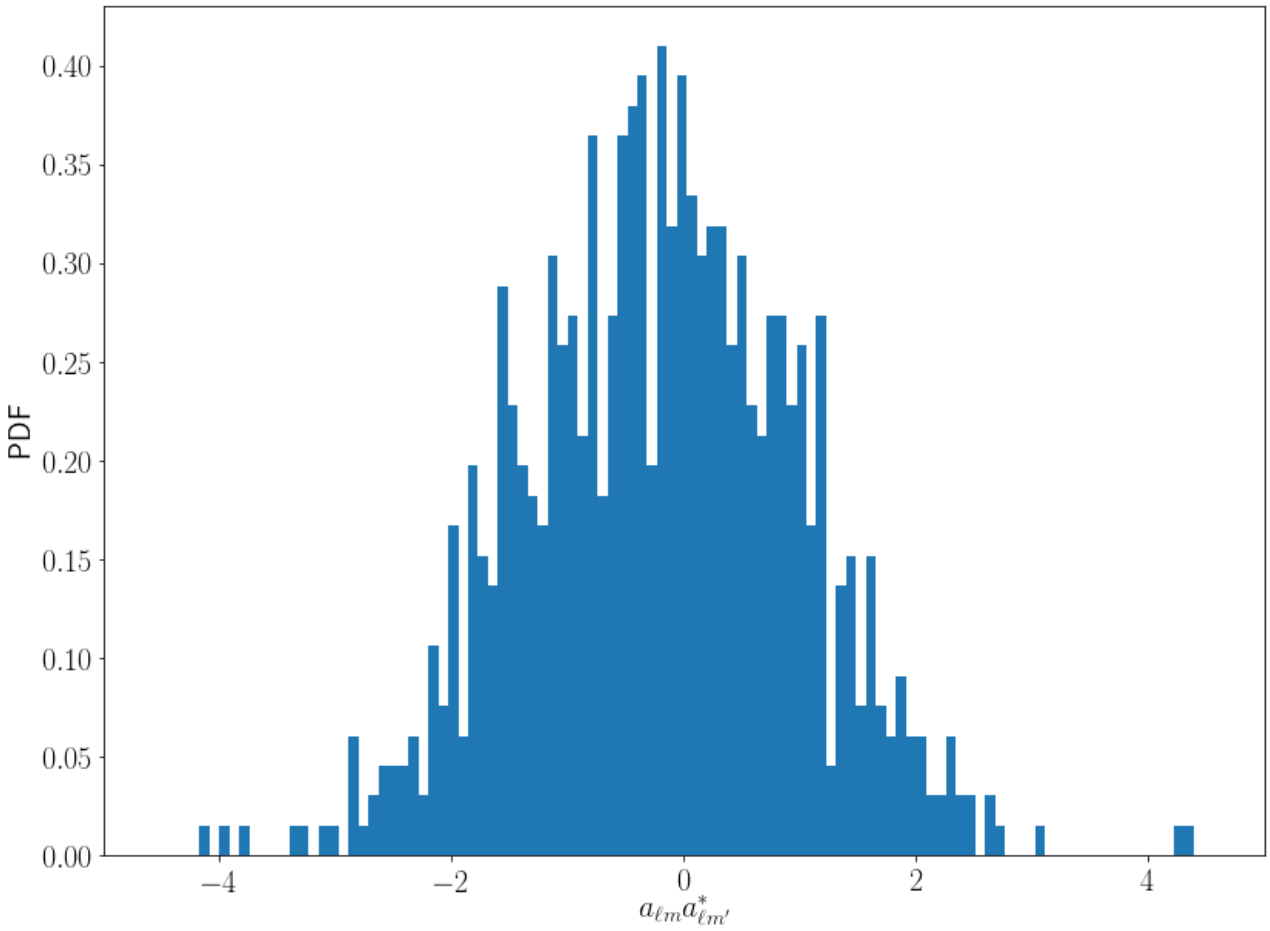
Luckily, `healpy` provides a couple of functions for this: `healpy.Alm.getidx` tells you the index in the array of a particular (ℓ, m), while `healpy.Alm.getlm` does the converse. Both require $\ell_{\rm max}$ as an argument, as this determines the order in which the $a_{\ell m}$'s are stored. You can find out the $\ell_{\rm max}$ of a given array using `healpy.Alm.getlmax`.

In [22]:
```
lmax = hp.Alm.getlmax(len(alm_syn))
idx = hp.Alm.getidx(lmax, 100, 50)
(l, m) = hp.Alm.getlm(lmax, 120839)

print("we have l_max={}".format(lmax))
print("the l=100, m=50 mode is stored at index={}".format(idx))
print("index 120839 contains the l={0}, m={1} mode".format(l, m))
```

```
we have l_max=767
the l=100, m=50 mode is stored at index=37225
index 120839 contains the l=656, m=177 mode
```

Exercise 3: For $\ell = 1500$, confirm that different m-modes are uncorrelated; i.e. check that for $m \neq m'$ the product $a_{\ell m} a^*_{\ell m'}$ is distributed around zero.

In [23]:
```
ell = 1500
asq = []

for m1 in range(ell+1):
    for m2 in range(m1):
        idx1 = hp.Alm.getidx(lmax, l, m1)
        idx2 = hp.Alm.getidx(lmax, l, m2)
        alm1 = alm_syn[idx1]
        alm2 = alm_syn[idx2]
        asq.append(alm1*alm2.conj())

fig = plt.figure()
fig.set_size_inches(12, 9)
plt.xlabel(r"$a_{\ell m}a^*_{\ell m'}$")
plt.ylabel("PDF")
plt.hist(std_errors, bins=100, density=True)
plt.xlim(-5.0, +5.0)
fig.tight_layout()
```

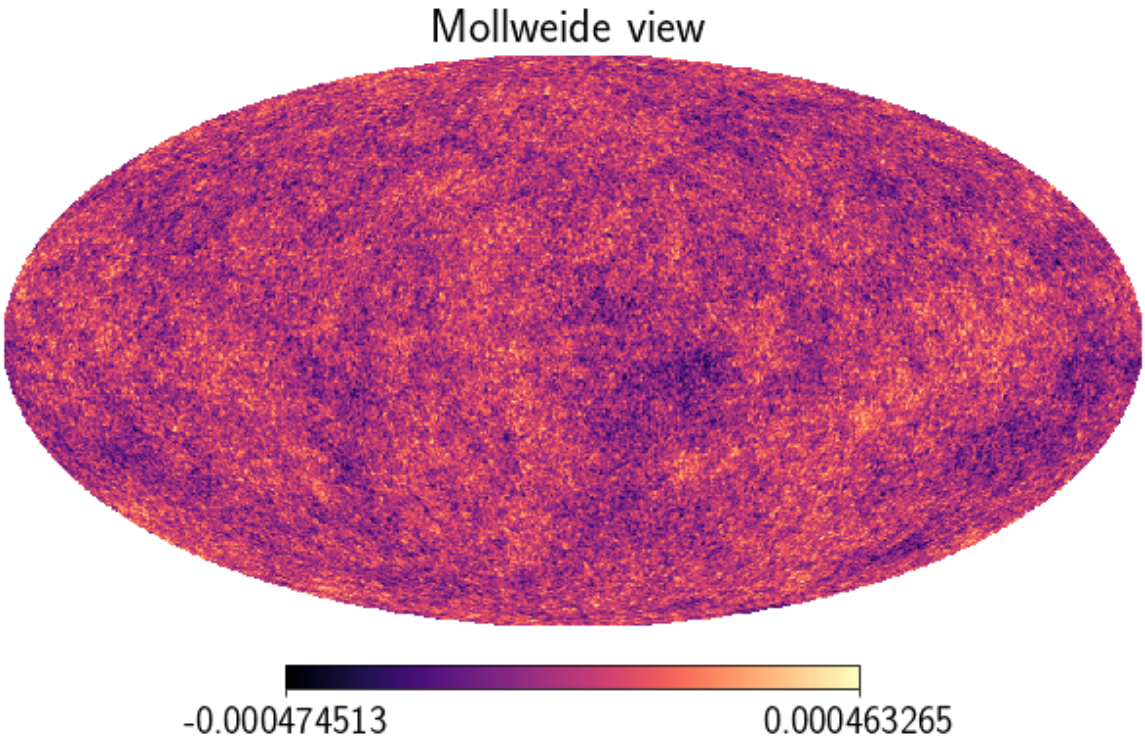


The real deal: Planck CMB maps

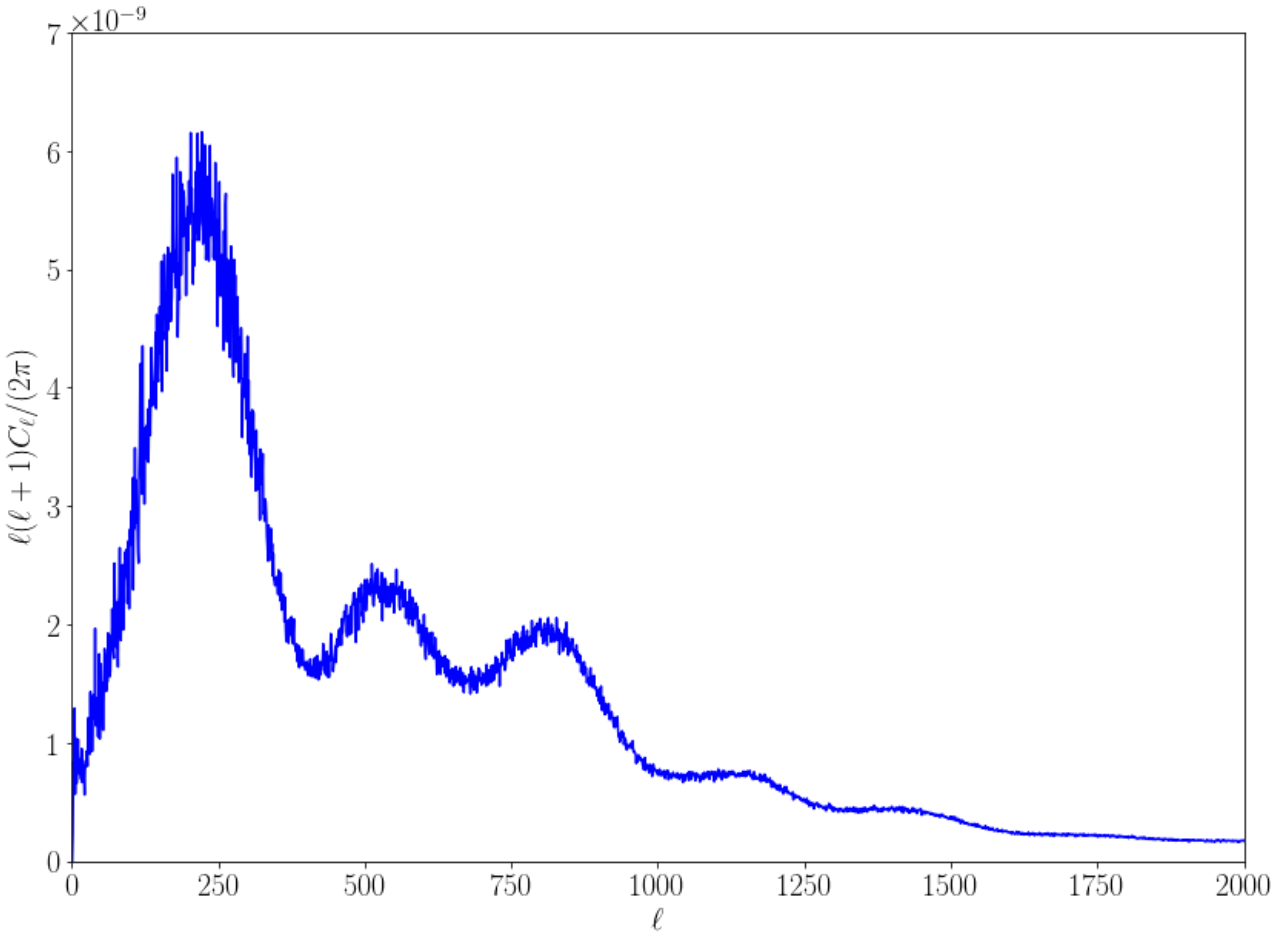
Let's download one of the Planck temperature maps and compute its spectrum. This map has a very high resolution ($\texttt{nside} = 2^{11}$), so this may take a little while.

In [24]:
```
#! wget -v https://irsa.ipac.caltech.edu/data/Planck/release_2/all-sky-maps/maps/component-maps/cmb/COM_CMB_IQU-commander-field-Int_2048_R2.01_full.fits
map_cmb = hp.read_map("COM_CMB_IQU-commander-field-Int_2048_R2.01_full.fits")
hp.mollview(map_cmb, cmap="magma")
```

```
NSIDE = 2048
ORDERING = NESTED in fits file
INDXSCHM = IMPLICIT

/home/alex/.local/lib/python2.7/site-packages/healpy/fitsfunc.py:372: UserWarning: No INDXSCHM keyword in header file : assume IMPLICIT
  warnings.warn("No INDXSCHM keyword in header file : " "assume {}".format(schm))

Ordering converted to RING
```

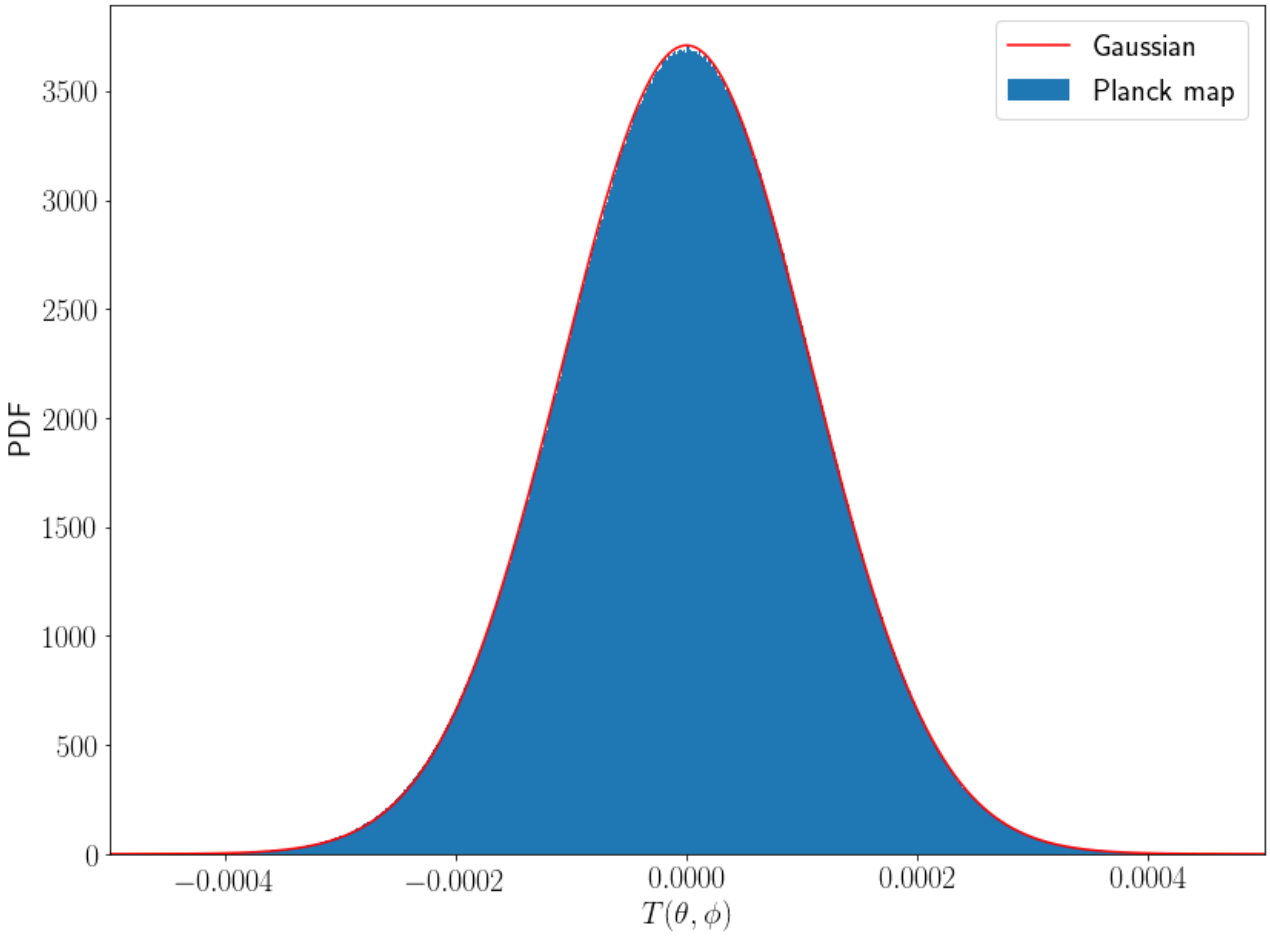


In [25]:
```
C_ell = hp.anafast(map_cmb, lmax=2000)
ell = np.arange(len(C_ell))
D_ell = np.array([0.5*l*(l+1.0)*C_ell[l]/np.pi for l in ell])

fig = plt.figure()
fig.set_size_inches(12, 9)
plt.xlabel(r"$\ell$")
plt.ylabel(r"$\ell(\ell+1)C_\ell/(2\pi)$")
plt.plot(ell, D_ell, "b")
plt.xlim(0, 2000)
plt.ylim(0.0, 7.0e-9)
fig.tight_layout()
```



Exercise 4: Plot a histogram of the pixel values to check that the CMB is Gaussian-distributed with zero mean and variance equal to $\sum_\ell (2\ell+1)C_\ell/(4\pi)$.

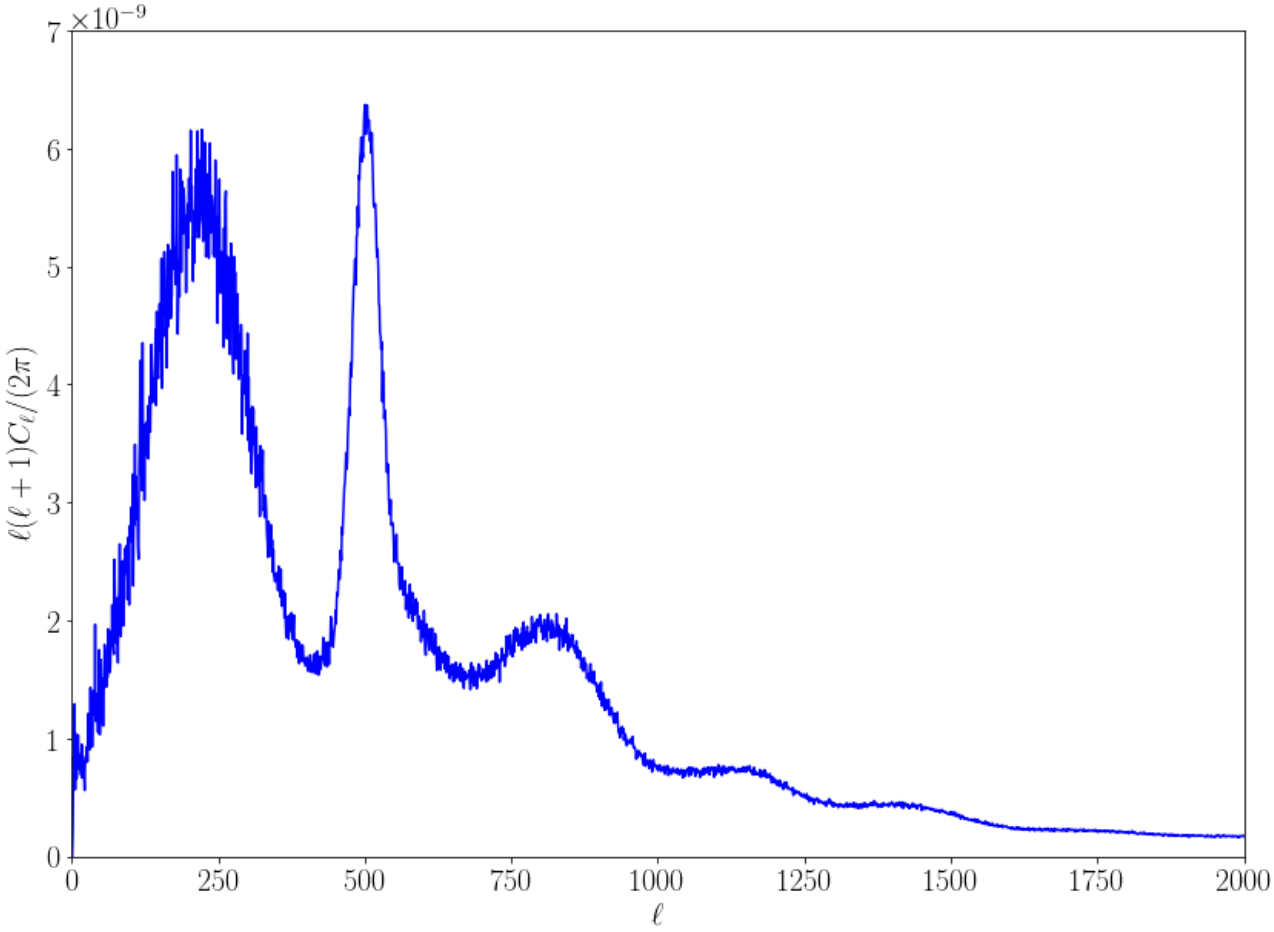
In [26]:
```
var = np.sum([0.25*(2.0*l+1.0)*C_ell[l]/np.pi for l in ell])

bins = np.linspace(-5.0e-4, +5.0e-4, num=1000)
gaussian = np.array([np.power(2.0*np.pi*var,-0.5)*np.exp(-0.5*bins[x]**2.0/
var) for x in xrange(1000)])

fig = plt.figure()
fig.set_size_inches(12, 9)
plt.xlabel(r"$T(\theta,\phi)$")
plt.ylabel("PDF")
n, bins, patches = plt.hist(map_cmb, bins=bins, density=True, label="Planck
map")
plt.plot(bins, gaussian, "r", label="Gaussian")
plt.xlim(-5.0e-4, +5.0e-4)
plt.legend(loc=0)
fig.tight_layout()
```

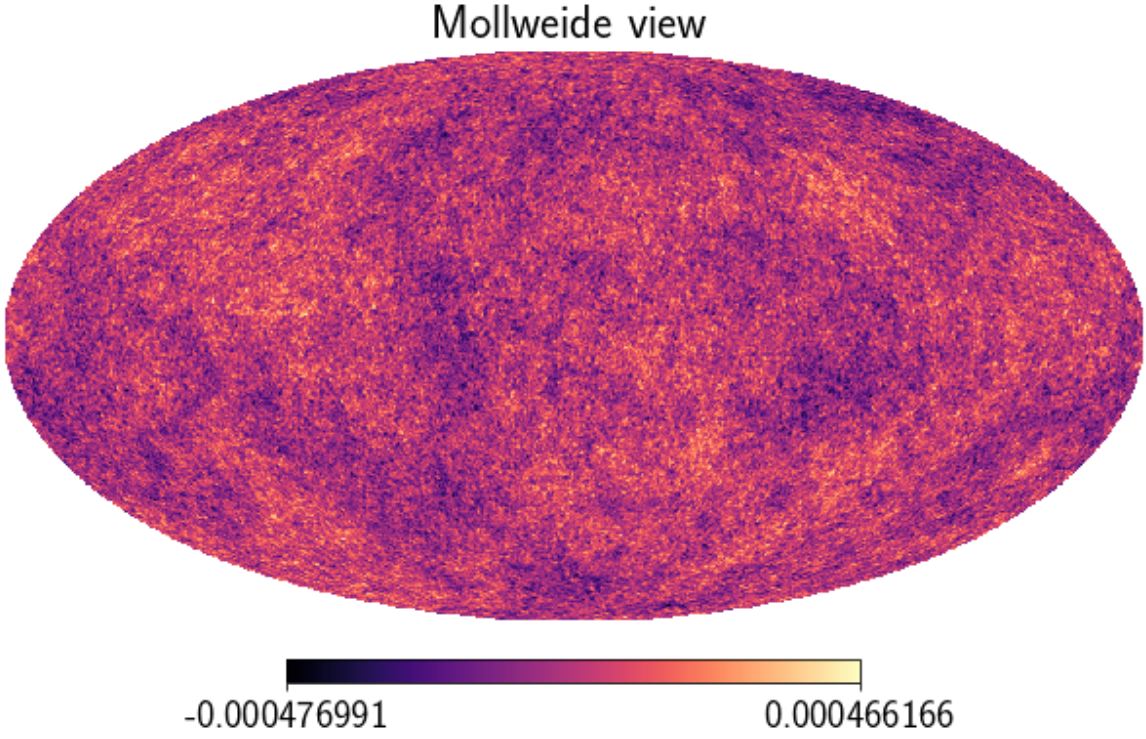


Exercise 5: Let's modify the Planck spectrum, and see how it changes the map. Add a new peak centred on $\ell = 500$, with a Gaussian profile with variance $\sigma_\ell = 500$, and height 1×10^{-13}.

In [27]:

```
new_peak = np.array([1.0e-13*np.exp(-0.5*(l-500.0)**2.0/500.0) for l in ell
])
C_ell_mod = C_ell + new_peak
D_ell_mod = np.array([0.5*l*(l+1.0)*C_ell_mod[l]/np.pi for l in ell])

fig = plt.figure()
fig.set_size_inches(12, 9)
plt.xlabel(r"$\ell$")
plt.ylabel(r"$\ell(\ell+1)C_\ell/(2\pi)$")
plt.plot(ell, D_ell_mod, "b")
plt.xlim(0, 2000)
plt.ylim(0.0, 7.0e-9)
fig.tight_layout()

map_mod = hp.synfast(C_ell_mod, nside, new=True, verbose=False)
hp.mollview(map_mod, cmap="magma")
```


$\times 10^{-9}$
$\ell(\ell+1)C_\ell/(2\pi)$
ℓ
0
1
2
3
4
5
6
7
0
250
500
750
1000
1250
1500
1750
2000


Mollweide view
-0.000476991
0.000466166