Homework 4

Problem 1

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import math
import sys
from scipy.stats import norm
```

# This is some custom code that I wrote
# You can pick it up from the website. (A link is given)
# http://hep.ucsb.edu/people/claudio/ph29-f23/ccHistStuff.py
sys.path.insert(0, '/Users/claudio/Dropbox/Claudio/ph29-w23/utils/')  # macOS
# sys.path.insert(0, 'C:\Users\Claud\Dropbox\Claudio\ph29-w23\utils\')  # Windows
from ccHistStuff import statBox

In [2]:
```python
# random seed
np.random.seed(232441)

N = int(1e6)

# r is like a N x 12 matrix
r = np.random.rand(12, N)

# Summing across columns
result = r.sum(axis=0)
```
In [3]: # Steal code from Exercise 2 of Discussion Session 5
# A reasonable bin size might be 100 bins, +/- 4 sigma around the min
nbins = 100
mean = 6
std = 1  # we are told to expect this
bins = np.linspace(mean-4*std, mean+4*std, nbins+1)

for i in [1,2]:
  ax = plt.subplot(210+i)
  _ = ax.hist(result,bins, alpha=0.5,label='sum rand 12')
  ax.grid()
  ax.set_xlim(bins[0],bins[-1])
  statBox(ax,result,bins,greek=True,fontsize='x-small',x=0.99)

  # g(x) = norm.pdf(x, loc=mean, scale=std) is the gaussian normalized to 1, ie,
  # Integral_minusInf_to_plusInf g(x)dx = 1
  # Let delta_x=binsize
  # We want to plot N*delta_x*g(x), and that should agree with the histogram
  x = np.linspace(bins[0],bins[-1],501)  # many points to make it smooth
  delta_x = bins[1]-bins[0]  # binsize
  ax.plot(x, N*delta_x*norm.pdf(x, loc=mean, scale=std), color='black',
           label='gauss function')
  ax.legend(loc='upper left',fontsize='x-small')
  if i==2:
    plt.yscale('log')
    ax.set_ylim(bottom=1)
  else:
    ax.set_ylim(bottom=0)
The sum of 12 uniformly distributed random numbers $r$ between 0 and 1 has mean 6 and is approximately gaussianly distributed with $\sigma_{\text{tot}} = 1$.

The Gaussian approximation is good to about $\pm 3\sigma_{\text{tot}}$.

Explanation:

The probability distribution function for a single $r$ is $p(r)\,dr = dr$.

This is properly normalized $\int_0^1 p(r)\,dr = 1$. The mean of $r$ is $\mu = \frac{1}{2}$.

The variance of a single random number $r$ is

$$\sigma^2 = E[(r - \mu)^2] = \int_0^1 (r - \frac{1}{2})^2 p(r)\,dr = \int_0^1 (r^2 - r + \frac{1}{4})\,dr$$

$$\sigma^2 = \frac{1}{3} - \frac{1}{2} + \frac{1}{4} = \frac{1}{12}$$

When adding $N$ independent (ie: uncorrelated) random numbers, the means and the variances add[1].

Mean $\mu_{\text{tot}} = \frac{N}{2}$. For $N = 12$, $\mu_{\text{tot}} = 6$.

Variance $\sigma_{\text{tot}}^2 = \frac{N}{12}$. For $N = 12$, $\sigma_{\text{tot}}^2 = 1$. 
Problem 2

In [9]: # The color map...pick whatever you like
     # More choices here
     # https://matplotlib.org/2.0.2/examples/color/
     thisMap = "jet"
     # thisMap = "brg"
     #thisMap = "terrain"
     #thisMap = "gnuplot"
     #thisMap="ocean"
     #thisMap ="brg"

In [4]: # The boundaries of the plot
     xmin = -1.6
     ymin = -1.
     xmax = 1.6
     ymax = 1.

     # the size of the figure in pixels
     xpix = 640
     ypix = 400

     # the x and y pixel increments
     dx = (xmax - xmin)/xpix
     dy = (ymax - ymin)/ypix

In [5]: # the pixel color map...instantiate a numpy array
     # with one entry per pixel and of unsigned integer
     # type, one byte (0 to 255)
     # (Actually, do not need to specify uint8)
     # pixColor = np.zeros((xpix,ypix), dtype=uint8)
     pixColor = np.zeros((xpix,ypix))
     maxPix = 255
In [6]:
# the constant
C = complex(-0.79, 0.156)

# Fill the pixColor array
for ix in range(xpix):
    x = xmin + ix*dx + 0.5*dx
    for iy in range(ypix):
        y = ymin + iy*dy + 0.5*dy
        thisIter = 0
        z = complex(x, y)
        while abs(z) < 2 and thisIter<maxPix:
            z = z*z + C
            thisIter = thisIter+1
        pixColor[ix,iy] = thisIter

In [7]:
# Stupid thing: when putting it on the screen
# the 1st index is the column and the 2nd index is the
# row. So in order to really have [ix, iy] we need to
# transpose before plotting.
# Even more stupid: the vertical dimension increases
# from the top, not from the bottom. The "flipud" function
# changes the order of the rows
newpixColor = np.flipud(pixColor.transpose())
Problem 3

In [11]: # read the data set
   # (Note: I downloaded it from the web and put it in my working directory)
   x = np.load("dataSet.npy")
In [12]: # Let's see how many entries, the largest, the smallest, the mean, the std
   print("There are ",len(x)," entries")
   print("Minimum = ", x.min())
   print("Maximum = ", x.max())
   print("Average = ", x.mean())
   print("Standard Deviation = ", x.std())

There are 50000 entries
Minimum = 1.5182138231403007e-05
Maximum = 23.88309149446749
Average = 2.2930107314269903
Standard Deviation = 2.281406451650508

Hmm....looks like most entries are around 2-3, but it extends all the way to ~ 25
Here is a 1st guess on how to plot it, we'll look at it and then adjust
In [13]:
# Don't bother to make it pretty yet...just exploring now
nbins = 50
bins = np.linspace(0, 25, nbins+1)
ax=plt.subplot(111)
_ = ax.hist(x, bins, histtype='step')

Doesn't look too bad, but I suspect that log scale would be better?
In [14]:

```python
ax = plt.subplot(111)
ax.hist(x, bins, histtype='stepfilled', color='red', log=True)
ax.set_xlabel('X')
ax.set_ylabel('Entries per 0.5')
statBox(ax, x, bins, y=0.95)
ax.set_xlim(0, 25)
# This is purely esthetics on the ticks (personal preference)
ax.tick_params("both", direction='in', length=8, right=True, top=True)
ax.set_xticks(bins, minor=True)
ax.tick_params("both", direction='in', length=4, right=True, which='minor', top=True)
ax.grid()
```

It looks like an exponential since with y in log scale it is linear vs. x