Physics IA - Data Analysis

Unknown Author

February 29, 2016

In [113]:
```
import math
import matplotlib as mpl
# mpl.use('pgf')
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from scipy import stats
from matplotlib import rc

%matplotlib inline

print plt.style.available
# plt.style.use('classic')
```

```
[u'seaborn-darkgrid', u'seaborn-notebook', u'classic', u'seaborn-
ticks', u'grayscale', u'bmh', u'seaborn-talk', u'dark_background',
u'ggplot', u'fivethirtyeight', u'seaborn-colorblind', u'seaborn-deep',
u'seaborn-whitegrid', u'seaborn-bright', u'seaborn-poster', u'seaborn-
muted', u'seaborn-paper', u'seaborn-white', u'seaborn-pastel', u
'seaborn-dark', u'seaborn-dark-palette']
```

In [114]:
```
## Change default plot style
def_style = 'bmh'
plt.style.use(def_style)
# plt.style.use('grayscale')
fig_scl = 3

def figsize(scale, height_ratio=(np.sqrt(5.0)-1.0)/2.0):
    # got after running \the\textwidth
    fig_width_pt = 345.000

    inches_per_pt = 1.0/72.0

    fig_width = fig_width_pt*inches_per_pt*scale
    fig_height = fig_width*height_ratio
    fig_size = (fig_width,fig_height)

    return fig_size

def setup_tex():
    """
    formats the plots for production with latex
    """
    pgf_with_latex = {
        "pgf.texsystem": "pdflatex",
        "text.usetex": True,
        "font.family": "serif",
        "font.serif": ["palatino"],
        "font.sans-serif": [],
```

```
        "font.monospace": [],
        "axes.labelsize": 10,
        "text.fontsize": 10,
        "legend.fontsize": 8,
        "xtick.labelsize": 8,
        "ytick.labelsize": 8,
        "figure.figsize": figsize(1)
    }
    mpl.rcParams.update(pgf_with_latex)

def savefig(filename):
    pass
#     plt.savefig('{}.pgf'.format(filename))
    plt.savefig('{}.pdf'.format(filename))
    plt.savefig('{}.eps'.format(filename))
```

In [115]:

```
## Load all required data
print "Loading data into pandas data frame..."
data_spring_coeff = pd.read_csv("p1_coeff.csv")
data_damped_motion = {k:pd.read_csv("DM" + str(k) + ".csv")
                          for k in [2, 3, 4, 5]}
print "Done loading."
```

```
Loading data into pandas data frame...
Done loading.
```

In [116]:

```
def get_intercept(m, (x, y)):
    """
    returns the y-intercept of a line given a
    single point <(x, y)> and the corresponding
    slope <m>
    """
    return (y - m*x)

def get_slope((x1, y1), (x2, y2)):
    """
    returns the slope of the line connecting the
    points <(x1, y1)> and <(x2, y2)>.
    """
    return float(y2 - y1)/float(x2 - x1)
```

1 Part I: Calculation of spring constant

We use the force (F) and position (x) data collected in the first part of the experiment to get the line of best fit for the x-F graph.

The slope of this line $\left(m = \dfrac{\Delta F}{\Delta x}\right)$ will give us $-k$ (since the position sensor was located at the bottom and therefore records $-x$ instead of x).

Hence,

$$-k = \frac{\Delta F}{\Delta x}$$

In [117]:

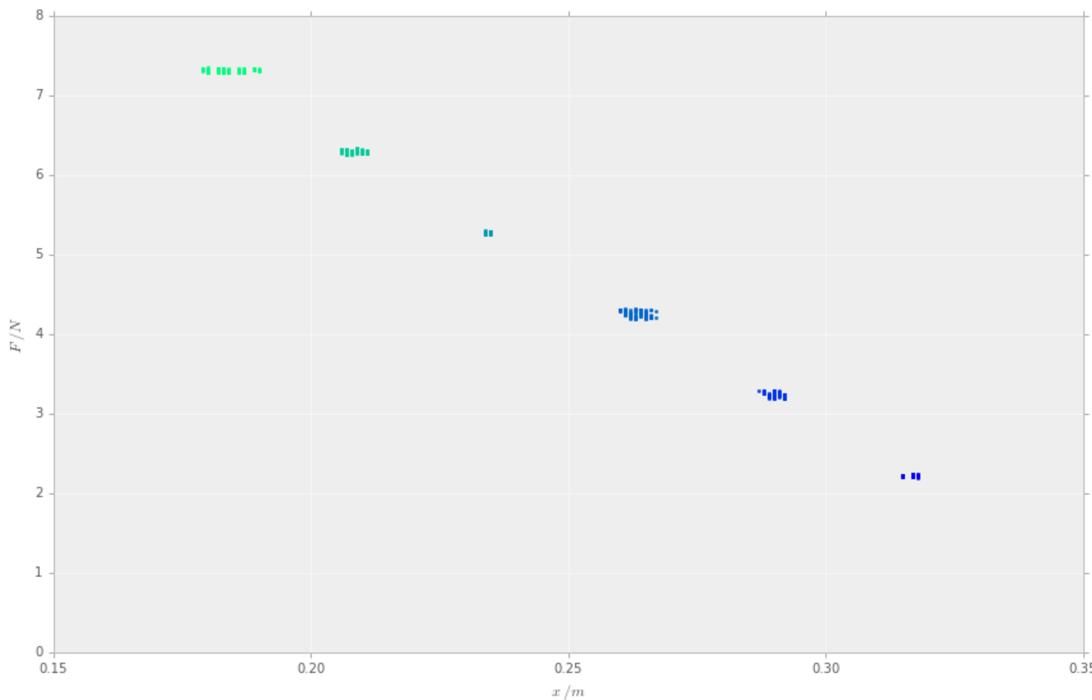
```
# store the Position (X) and Force (F) data in arrays
X_rw = np.array([data_spring_coeff['P M' + str(k)]
                     for k in range(6)])
F_rw = np.array([data_spring_coeff['F M' + str(k)]
                     for k in range(6)])

# calculate the mean value and related uncertainties
# (standard deviation)
X = [(x.mean(), x.std()) for x in X_rw]
```

```
F = [(f.mean(), f.std()) for f in F_rw]

print X
```

```
[(0.31733200000000006, 0.00073059975362711489), (0.29065999999999997,
0.0011799999999999983), (0.26306000000000002, 0.0017596590578859317),
(0.23413199999999995, 0.00033849076796864082), (0.20786000000000002,
0.0013741906709041499), (0.18293199999999998, 0.0025122452109617032)]
```

In [118]:
```
## Plot the raw data points
plt.clf()
plt.figure(figsize=figsize(fig_scl))

plt.xlim([0.15, 0.35])
plt.ylim([0.0, 8.00])

plt.xlabel('$x\; /m$')
plt.ylabel('$F\; /N$')

c = cm.winter(np.linspace(0, 1, len(X_rw)))

for (x_p, f_p, c) in zip(X_rw, F_rw, c):
    plt.scatter(x_p, f_p, marker='.', color=c)
savefig('pt1_x-F_raw_graph')
```

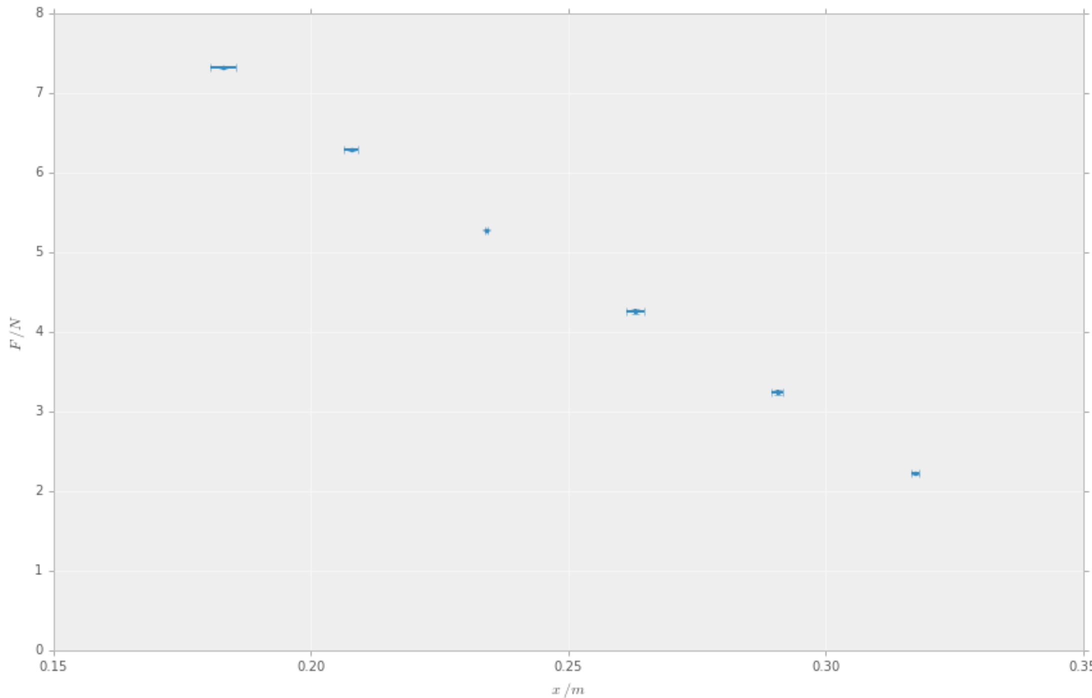
```
<matplotlib.figure.Figure at 0x7fca0be57a10>
```



In [119]:
```
# Plot the x-F graph with error bars.
# Also, save the file with a proper name.
plt.clf()
plt.figure(figsize=figsize(fig_scl))

plt.xlim([0.15, 0.35])
plt.ylim([0.0, 8.00])

plt.xlabel('$x\; /m$')
plt.ylabel('$F\; /N$')
```

```python
plt.errorbar([x[0] for x in X],
             [f[0] for f in F],
                 xerr=[x[1] for x in X],
                 yerr=[f[1] for f in F],
             fmt='--.', linestyle="None")

savefig('pt1_x-F_graph')
```

<matplotlib.figure.Figure at 0x7fca0bcffd90>



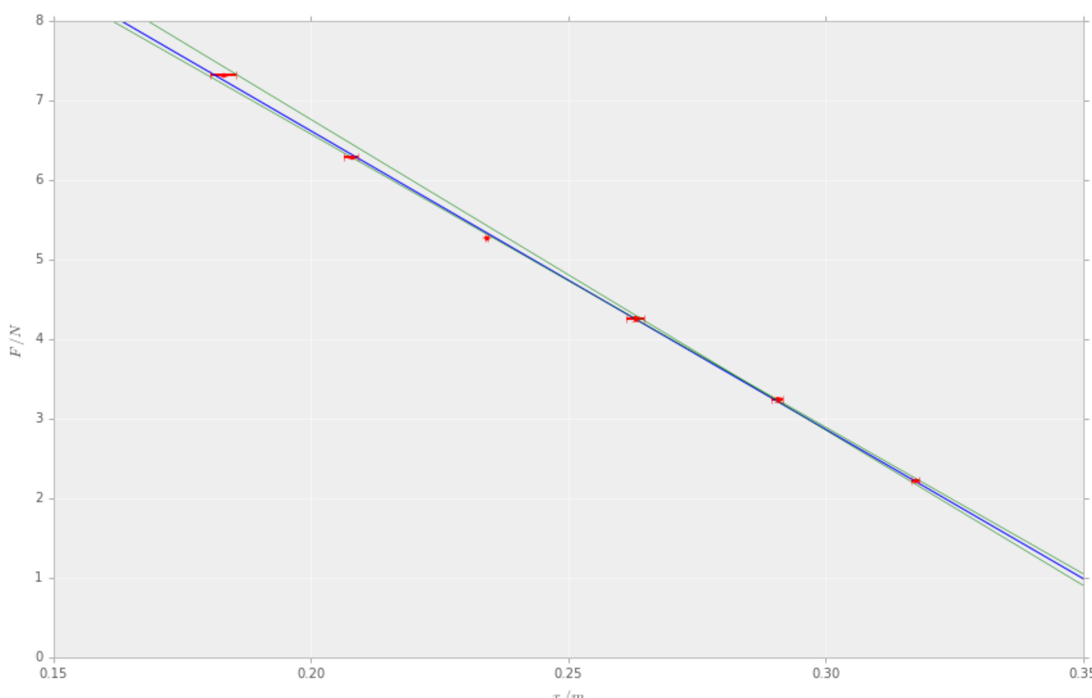
In [120]:
```python
## Get the line of best fit for the x-F graph
m, b = np.polyfit(X_rw.flatten(), F_rw.flatten(), 1)

## Get the other bounding lines for the main line
m_1 = get_slope((X[0][0]-X[0][1],
                 F[0][0]-F[0][1]),
                (X[-1][0]+X[-1][1],
                 F[-1][0]+F[-1][1]))

b_1 = get_intercept(m_1, (X[0][0]-X[0][1],
                          F[0][0]-F[0][1]))

m_2 = get_slope((X[0][0]+X[0][1],
                 F[0][0]+F[0][1]),
                (X[-1][0]-X[-1][1],
                 F[-1][0]-F[-1][1]))

b_2 = get_intercept(m_2, (X[0][0]+X[0][1],
                          F[0][0]+F[0][1]))

## Print out the values
print "Line of best fit:"
print "y = {}x + {}".format(m, b)
print ""
print "Lines of min/max slope:"
print "* y = {}x + {}".format(m_1, b_1)
print "* y = {}x + {}".format(m_2, b_2)
```

```
Line of best fit:
y = -37.5365286233x + 14.1271896573

Lines of min/max slope:
* y = -39.0690034548x + 14.5782117522
* y = -36.8643692166x + 13.9526105591
```

In [121]:

```
# Plot the lines of best and worst fit.
plt.clf()
plt.figure(figsize=figsize(fig_scl))

plt.xlim([0.15, 0.35])
plt.ylim([0.0, 8.00])

plt.xlabel('$x\; /m$')
plt.ylabel('$F\; /N$')

xs = np.linspace(0.15, 0.35, 10)

plt.plot(xs, m*xs+b, 'b-', linewidth=1)
plt.plot(xs, m_1*xs+b_1, 'g-', linewidth=0.5)
plt.plot(xs, m_2*xs+b_2, 'g-', linewidth=0.5)

plt.errorbar([x[0] for x in X],
             [f[0] for f in F],
             xerr=[x[1] for x in X],
             yerr=[f[1] for f in F],
             fmt='r.',
             linestyle="None",
             ecolor='r')

savefig('pt1_x-F_graph_fit_lines')
```

```
<matplotlib.figure.Figure at 0x7fca0bcff3d0>
```



In [122]:
```
## Calculation of coefficient of friction and the related
## uncertainties
k = -m
k_1 = -m_1
k_2 = -m_2

k_err = (k_1 - k_2)/2.0

spring_coeff = k
spring_coeff_err = k_err

print "Spring coeff."
print "Best:\t{} N/m".format(round(k, 3))
print "Max:\t{} N/m".format(round(k_1, 3))
print "Min:\t{} N/m".format(round(k_2, 3))
print "A. Err:\t{} N/m".format(round(k_err, 2))
print "P. Err:\t{}%".format(round((k_err/k)*100, 2))
```

```
Spring coeff.
Best:   37.537 N/m
Max:    39.069 N/m
Min:    36.864 N/m
A. Err: 1.1 N/m
P. Err: 2.94%
```

2 Part 2: Calculation of Damping coeff.

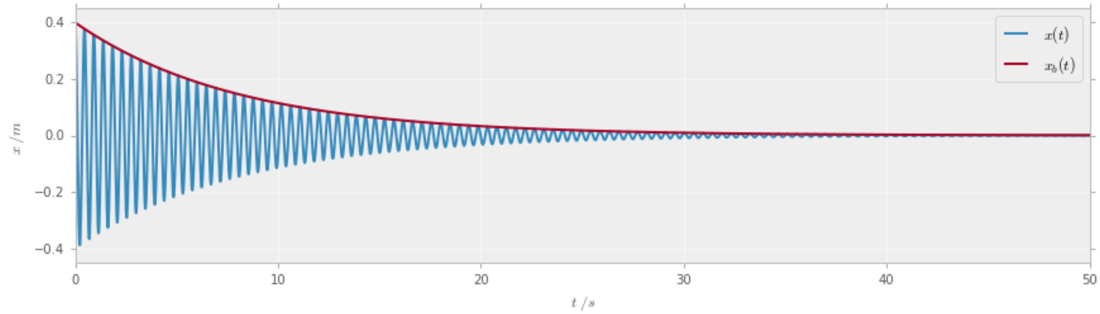
In a damped simple harmonic motion, the location x of the particle is given by:

$x(t) = Ae^{\frac{-bt}{2m}} \cos{(\omega' t + \phi)}$

Where * A is the initial alplitude * m is the mass of the body * b is the damping coeff. * $\omega' = \sqrt{\omega^2 - \gamma^2} = \sqrt{\frac{k}{m} - \frac{b^2}{4m^2}}$

In [124]:
```
## Damped simple harmonic motion: EXAMPLE
def ex_damped_shm(A, m, b, k, ph = 0):
    t = np.linspace(0, 50, 10000)
    gamma = float(b)/float(2*m)
    omega = math.sqrt(float(k)/float(m))
    omega_prime = math.sqrt(omega**2 - gamma**2)
    x = []
    for T in t:
        trig_coeff = (math.cos(omega_prime*T + ph))
        exp = ((-1*T*float(b))/float(2*m))
        x.append(A*(math.e**exp)*trig_coeff)

    x_e = A*(math.e**((-1*t*float(b))/float(2*m)))

    plt.plot(t, x, label="$x(t)$")
    plt.plot(t, x_e, label="$x_b(t)$", linewidth=2)
    plt.legend()

plt.clf()
plt.figure(figsize=figsize(fig_scl, height_ratio=(1.0/4.0)))
plt.xlim([0, 50])
plt.xlabel('$t\;/s$')
plt.ylim([-0.45, 0.45])
plt.ylabel('$x\;/m$')
ex_damped_shm(0.4, 0.2, 0.05, 37)

savefig("ex_damped")
```

<matplotlib.figure.Figure at 0x7fca0c6ef190>



2.1 Change of Phase

Since all data points have different values for the phase angle, ϕ, for the sake of convinience, we will bring all the phases to 0 which will then allow us to take averages, etc.

Since the F value has the most consistent readings (least number of outliers), we shall use that to find the phase angle and then bring it to 0.

In [123]: