Error propagation: Three approaches

- H&H Functional Approach
- H&H Calculus Approach
- Monte Carlo simulation

In [33]:
```
import scipy as sp
from scipy import stats
import sympy as sym # for symbolic differentiation
from sympy.interactive import init_printing # provides LaTex formatted outpu
sym.init_printing()
```

In PHYS 211 we did an experiment to determine "little g" from measurement of the length and period of a pendulum. The following cell defines a function giving g for known values of l and T; the next cell it gives typical data and associated uncertainties.

In [8]:
```
def g(l,t):
    return 4*sp.pi**2*l/t**2
```

In [9]:
```
mean_l = 1.0    # Measured length of pendulum in meters
sigma_l = 0.004 # Uncertainty in length - assumed to be
                #i.d. of normal distribution
mean_t = 2.0;   # Measured period
sigma_t = 0.005 # Uncertainty in  period - assumed to be
                # s.d. of normal distribution
```

"Functional approach"

$$\begin{aligned}(\sigma_g)_L &= g(L+\Delta L, T) - g(L,T)\\ (\sigma_g)_T &= g(L, T+\Delta T) - g(L,T)\\ \sigma_g &= \sqrt{(\sigma_g)_L^2 + (\sigma_g)_T^2}\end{aligned}$$

In [10]:
```
sigma_gl = g(mean_l+sigma_l,mean_t)-g(mean_l,mean_t)
sigma_gt = g(mean_l,mean_t+sigma_t)- g(mean_l,mean_t)
sigma_g = sp.sqrt(sigma_gl**2 + sigma_gt**2)
g(mean_l,mean_t),sigma_gl,sigma_gt,sigma_g
```

Out[10]: $(9.869604401089358,\quad 0.03947841760435722,\quad -0.04916358185130853 5,\quad 0.0$

Result: $g = 9.87 \pm 0.06$

"Calculus approach"

$$
\begin{aligned}
(\sigma_g)_L &= \left.\frac{\partial g}{\partial L}\right|_{\bar{L},\bar{T}} \sigma_L \\
(\sigma_g)_T &= \left.\frac{\partial g}{\partial T}\right|_{\bar{L},\bar{T}} \sigma_T \\
\sigma_g &= \sqrt{(\sigma_g)_L^2 + (\sigma_g)_T^2}
\end{aligned}
$$

Redefine $g(L, T)$in terms of sympy floats so that we can do symbolic work. Will also use sym.sqrt() below.

NOTE: sympy floats != regular python floats See http://docs.sympy.org/dev/gotchas.html (http://docs.sympy.org/dev/gotchas.html) Either sym.sqrt(sym.pi), or sp.sqrt(float(sym.py)), or sym.sqrt(sym.Float(sp.pi)), or sym.sqrt(sym.syimpify(sp.pi))

In [24]:
```
def g(l,t):
    return 4*l*sym.pi**2/t**2
```

In sympy you must declare symbolic variable explicitly.
The method below allows you to control assumptions, as in

n = sym.symbols('n',postive=True)

You can also import directly from a set of common symbols, e.g.,

from sympy.abc import w or sym.var('z')

see http://docs.sympy.org/latest/gotchas.html#symbols (http://docs.sympy.org/latest/gotchas.html#symbols)

In [34]:
```
l, t = sym.symbols('l, t')
```

In [37]:
```
sym.diff(g(l,t),t,1),sym.diff(g(l,t),l)
```

Out[37]:

$$\left(-\frac{8l}{t^3}\pi^2, \quad \frac{4\pi^2}{t^2}\right)$$

Evaluate the symbolic expressions at the values $\bar{l}$ and $\bar{t}$, and calculate the uncertainty.

In [38]:
```
sigma_gt = sym.diff(g(l,t),l).evalf(subs={l:mean_l,t:mean_t})*sigma_t
sigma_gl = sym.diff(g(l,t),l).evalf(subs={l:mean_l,t:mean_t})*sigma_l
sigma_g = sym.sqrt(sigma_gl**2 + sigma_gt**2).evalf()
g(mean_l,mean_t).evalf(),sigma_gl,sigma_gt,sigma_g
```

Out[38]:

(9.86960440108936, 0.0394784176043574, 0.0493480220054468, 0.0631963

Result: $g = 9.87 \pm 0.06$

This agrees with previous result.

Monte Carlo technique

Here we are going to get the uncertainty in a different way, using simulated data. Our propagation of uncertainty formulas assume that the uncertainties are standard deviations of normal distribution. We can "redo" the experment, with new simulated "measurements" sampled from the assumed distributions.

First we return to our definition of $g(L,T)$that returns a regular python float.

In [39]:
```
def g(l,t):
    return 4*sp.pi**2*l/t**2
```

In [40]:
```
nEx = 10000                               # Number of simulated experiments
l = sp.random.normal(mean_l,sigma_l,nEx)  # Pick values of l from normal dis
t = sp.random.normal(mean_t,sigma_t,nEx)  # Pick values of t from normal dis
gg = g(l,t)                               # Calculate nEx values of g using
gg
```

Out[40]:
```
array([ 9.9317706 ,  9.89335535,  9.87379594, ...,  9.79706201,
        9.89475928,  9.93274119])
```

In [41]:
```
sp.mean(gg),sp.std(gg)
```

Out[41]: $(9.86959434768,\quad 0.0633946166306)$

Result: $g = 9.87 \pm 0.06$

In [42]:
```
%load_ext version_information
%version_information numpy, sympy
```

Out[42]:

Software	**Version**
Python	3.5.1 64bit [GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
IPython	4.2.0
OS	Linux 3.10.0 327.el7.x86_64 x86_64 with redhat 7.2 Maipo
numpy	1.11.0
sympy	1.0
Thu Sep 15 13:44:35 2016 EDT	

In [ ]:
```

```