Assignment 1

Brandon Lampe

ME-500: Numerical Methods in Mechanical Engineering

08/31/15

1 Summary of Relevant Theory

A numerical routine was written to approximate the definite integral of a function based on the Trapezoidal rule, and a convergence analysis was then performed. However; I am unsure of how to compare the theoretical rate of convergence to the numerical rate I computed. Additionally, I was unable to write a working Gauss quadrature routine; therefore, an available code was used to perform the Gauss quadrature of a third order polynomial.

2 Program General Trapezoidal Rule

The general trapezoidal rule (Equation 1) was utilized to calculate an numerical approximation to the definite integral (quadrature) of an arbitrarily chosen Function 2.

$$I_{num} = \alpha f(a) + (1-\alpha)f(b), \tag{1}$$

$$f(x) = sin(x) + x \tag{2}$$

The problem domain, $x \in [0, 10]$, was divided into 15 subdomains, where a and b in Equation 1 represent the upper and lower bounds of each subdomain, respectively. The analytical solution to the definite integral was calculated (Equation 3) and quadrature estimates with $\alpha = 0, 0.5, 1$ resulted in values of $I_{num} = 48.62, 51.77, 54.92$, respectively. Additionally, approximations for each subdomain are shown in Figure 1, where each point is centered at the horizontal midpoint of its respective subdomain.

$$\int_0^{10} sin(x) + x \, \mathrm{d}x = 51.84 \tag{3}$$

These results show that the trapezoidal rule most closely approximates the function when $\alpha = 0.5$, and quadrature results for $\alpha = 0$ and 1 proivde approximately the same error but on opposite sides of the analytical solution. The Python code for this algorithm is shown in the attached code listing.

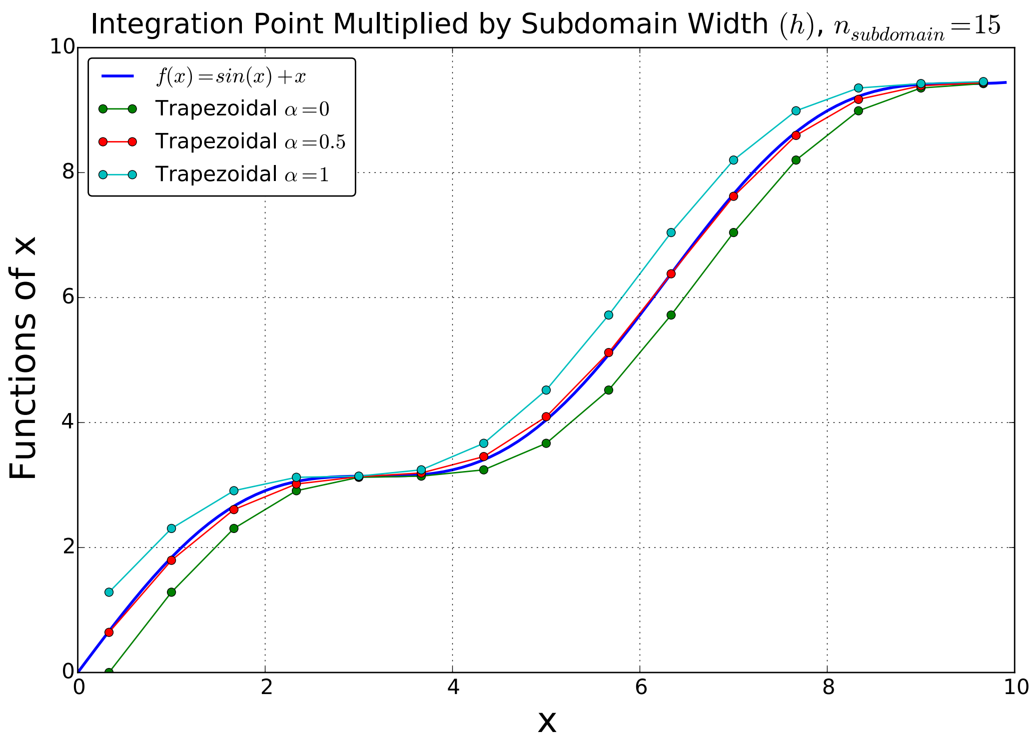


Figure 1: Results of the approximation to Equation 2.

3 Determine the Rate of Convergence

The same function (Equation 2) was chosen with the domain of, $x \in [0, 10]$, and the exact integral to this function over the defined domain was calculated as $I_{ann} = 51.84$. Using this value, the error was defined as:

$$Error = |I_{ann} - I_{num}|$$

The error was calculated over a range of subdomains ranging from 1 to 64, with the corresponding suddomain legth (h) ranging from 10 down to 0.16. Figure 2 shows the results of the convergence analysis. The highest rate of convergence clearly occurs when $\alpha = 0.5$ with a value of approximately 2, the convergence rate for $\alpha = 0$ and 1 were approximately unity (values shown on Figure 2).

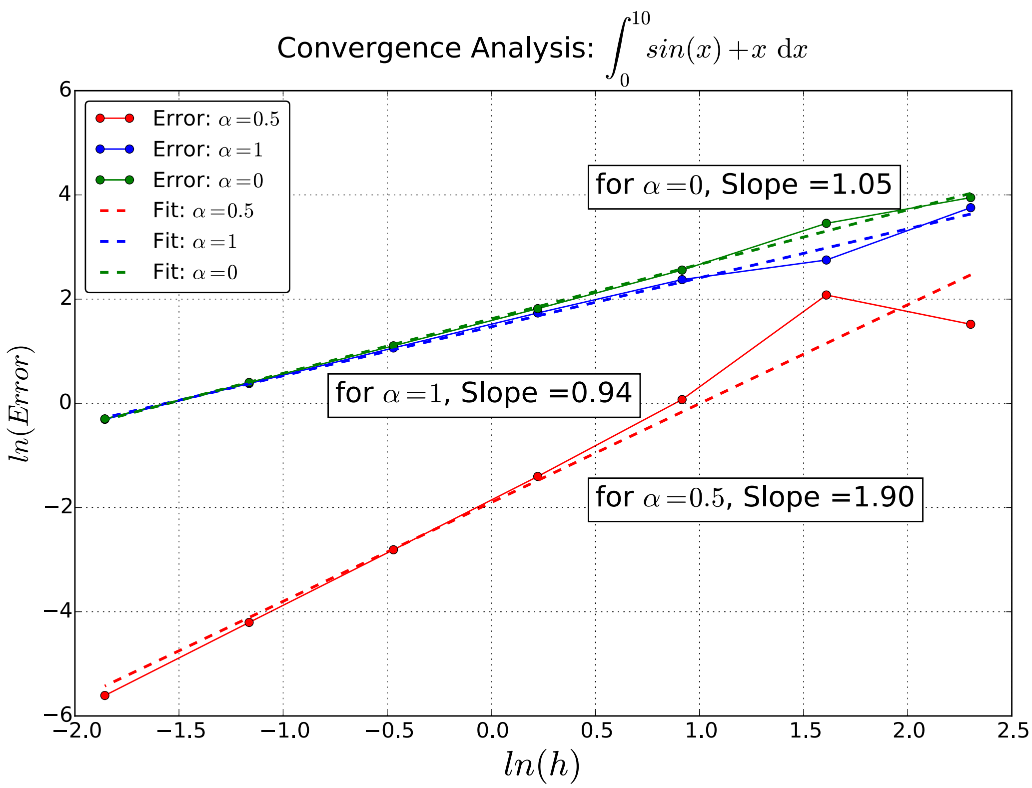


Figure 2: Convergence analysis.

I am unsure of how to compare the numerically calculated rate of convergence (slopes listed on plot) to the theoretical values. This is because the theoretical upper bound for the rate of convergence was said to be of the order of the polynomial function plus one ($n_p + 1$); however, the function I ingetrated was not a polynomial.

4 Gauss Quadrature

Third, fourth, fifth, and sixth-order taylor series expansions (TSE) of Equation 2 about $x = 5$ were performed to obtain the functions shown in Figure 3. The polynomial obtained from the third-order expansion is shown in Equation 4.

$$f(x) = -0.047x^3 + 1.189x^2 - 7.056x + 15.12 \quad (4)$$

I wrote a third-order Gauss quadrature routine ($GaussQuad\_3$) and evaluated the error associated each of the different TSE polynomials over a range of subdomain widths. Results of analysis hown in Figure4. These results show that a negligible amount of error was calculated when my third-order quadrature routine evaluated a third-order polynomail, hence the rate of convergence was zero. For higher order polynomials (fourth, fifth, and sixth), the rate of convergence was appears to be four, but when calculated these values were 3.6 and 2.4 for fifth and sixth-order polynomials, respectively.

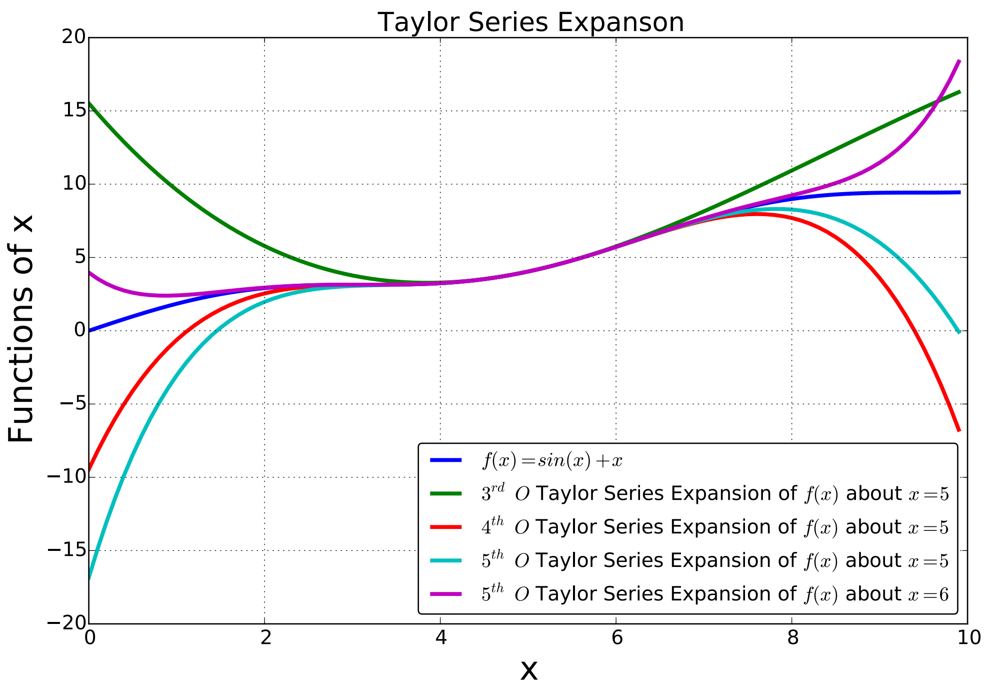


Figure 3: Taylor series expansions of Equation 2.

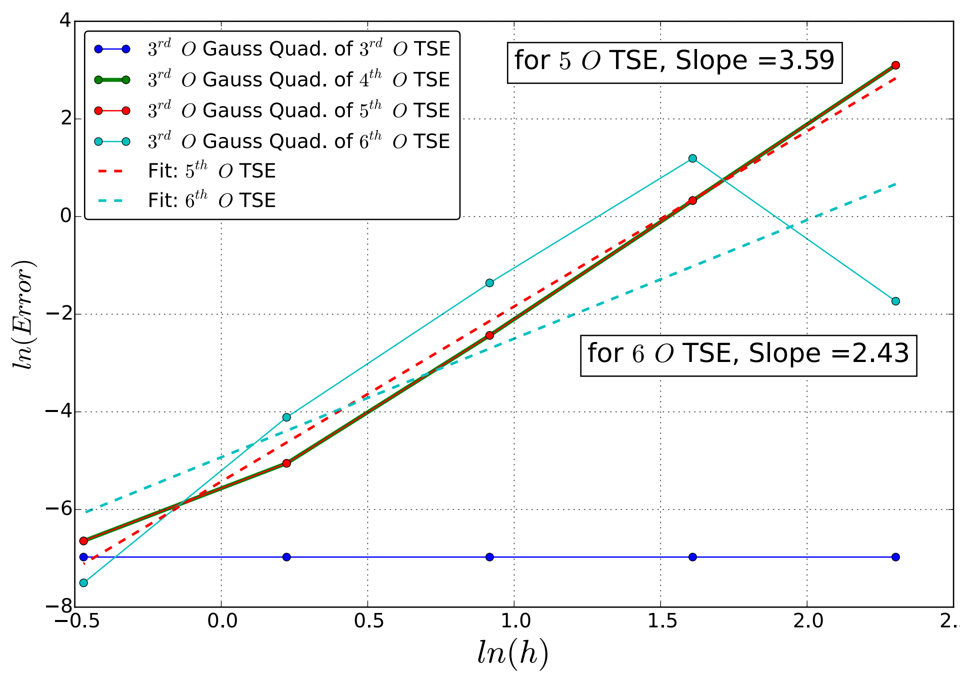


Figure 4: Analysis of Gauss quadrature of Equation 4.

In [1]:
```
from pylab import *
import numpy as np
import scipy as sp
from scipy.integrate import quad
import math
from matplotlib import pyplot as plt
from IPython.display import display

np.set_printoptions(precision = 3)
%matplotlib inline
# %pdb
# %pylab
```

Taylor Series Analysis

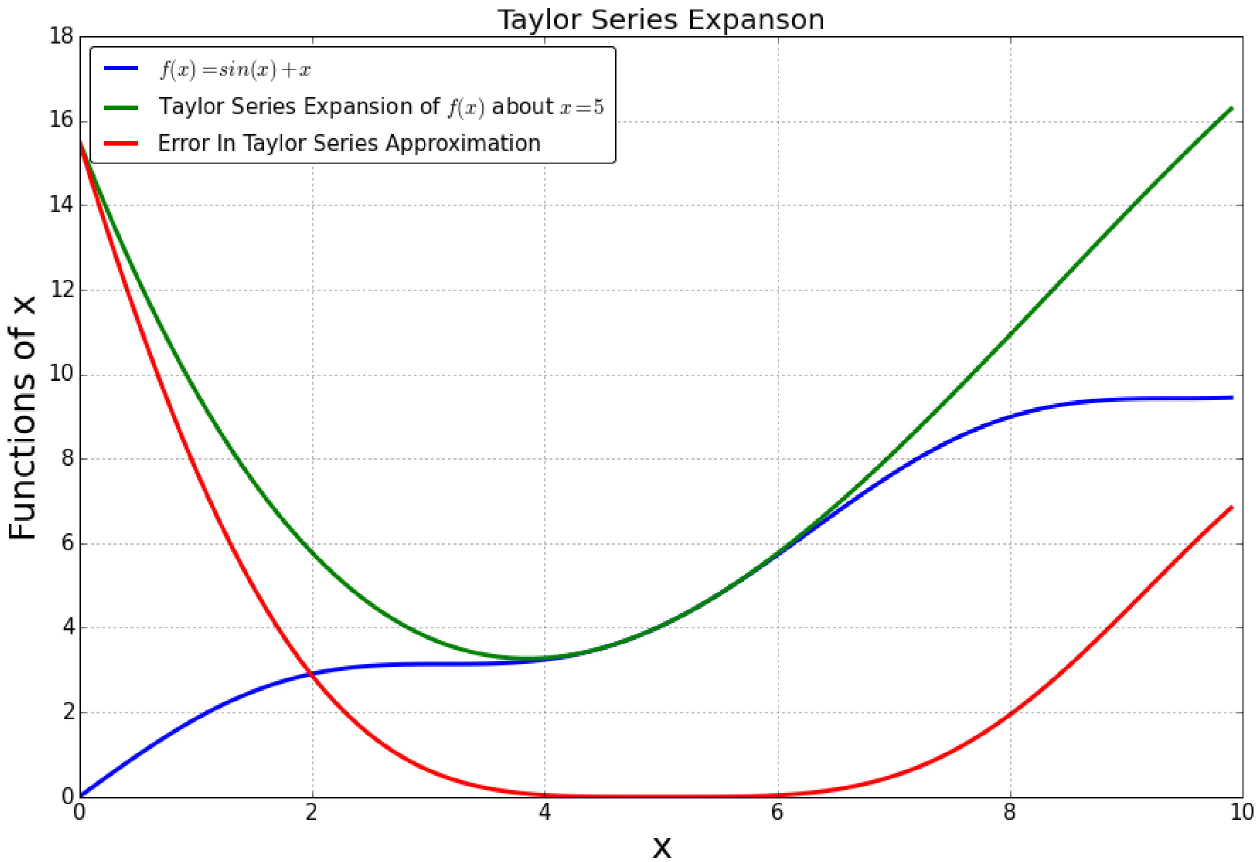
Define Function and Derivatives

In [2]:
```
# define functions and derivatives wrt x
def f(x):# function
    out = np.sin(x) + x
    return out

def f_p1(x):#dx
    out = np.cos(x) + 1
    return out

def f_p2(x):#d2x
    out = -np.sin(x)
    return out

def f_p3(x): #d3x
    out = -np.cos(x)
    return out
```

Define Domain and Taylor Series Expansion (TSE) about point "a"

In [3]:
```
# define the domain
x_min = 0
x_max = 10
x_delta = 0.1
x = np.arange(x_min, x_max, x_delta)

# the function
f_x = f(x)

# Taylor Series Expansion (third order polynomail)
a = 5 # about this value
TSE_a = f(a) + f_p1(a) / math.factorial(1) * (x - a)**(1) + f_p2(a) / m
ath.factorial(2) * (x - a)**2 + (
    f_p3(a) / math.factorial(3) * (x - a)**3)

# error in taylor series
TSE_E = np.abs(f(x) - TSE_a)
```

In [4]:
```
f_01, (ax1) = plt.subplots(1,1, sharex = True, sharey = True, figsize=
(12,8))
f_name = "01_TaylorSeries.pdf"

# plot
ax1.plot(x,f_x,lw = 3, label = ''r'$f(x)=sin(x)+x$')
ax1.plot(x,TSE_a,lw = 3, label = 'Taylor Series Expansion of 'r'$f(x)$
about $x=5$')
ax1.plot(x,TSE_E,lw = 3, label = 'Error In Taylor Series Approximatio
n')

ax1.grid(b = True, which = 'minor')
ax1.grid(b = True, which = 'major')
plt.tight_layout()
ax1.legend(loc=0, fontsize = 15, framealpha = 1, fancybox = True)
ax1.set_xlabel(''r'x', fontsize = 25)
ax1.tick_params(axis = 'x', labelsize = 15)
ax1.tick_params(axis = 'y', labelsize = 15)

# ax.set_ylim(4, 9)
# # ax.set_xscale('log')
# # ax.set_yscale('log')
# plt.rcParams['font.size']=20


ax1.set_ylabel(''r'Functions of x', fontsize = 25)

ax1.set_title(''r'Taylor Series Expanson', fontsize = 20)

path = "/Users/Lampe/Documents/UNM_Courses/ME-500/HW01"
out_file = path + "/" + f_name
f_01.savefig(out_file)
```



Prob 2:

Transformation and Integration Functions

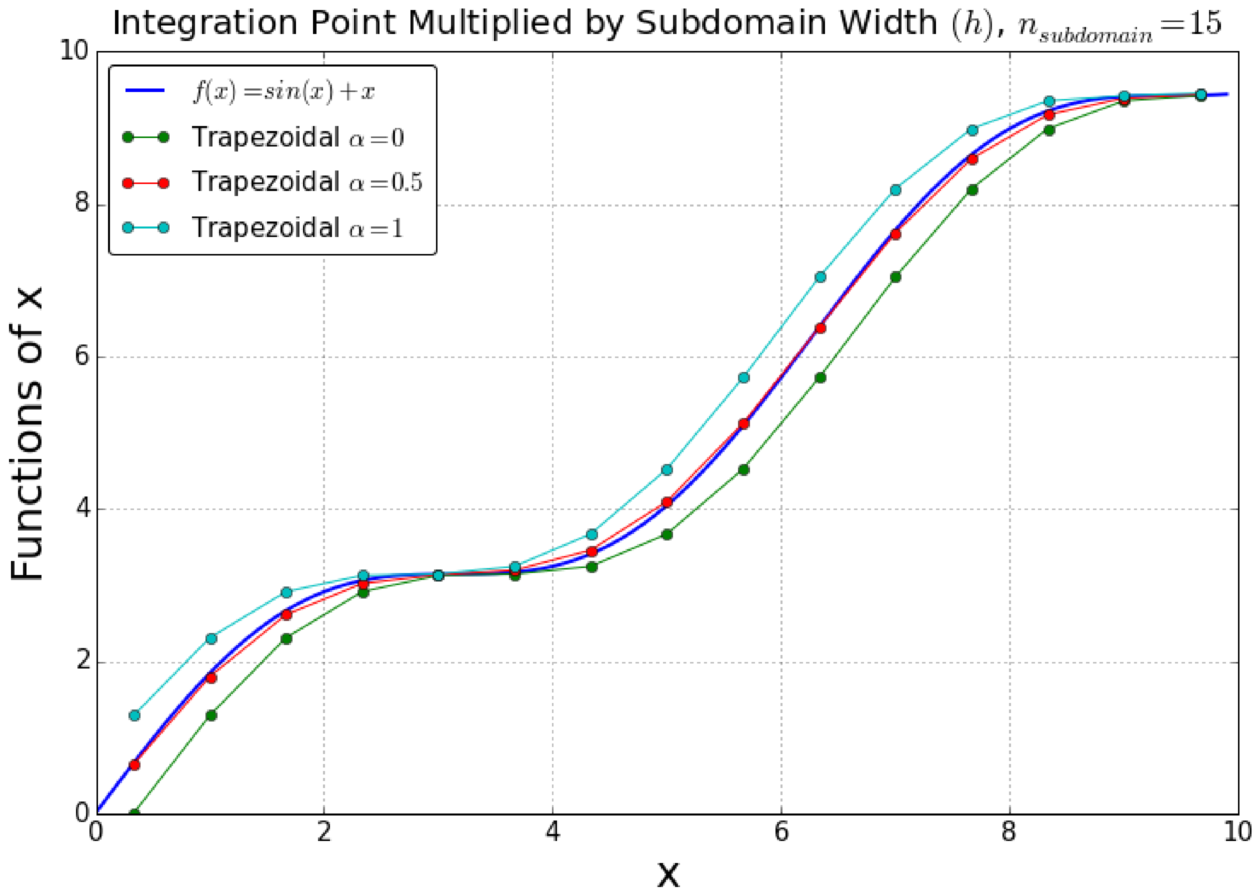
In [5]:
```
def Domain_Trans(val, lower_bound, upper_bound, normalize):
    """transforms a value from a domain ranging from 0 -> 1 to the valu
e in an arbitrary domain and vice versa
    val: scalar value to be transformed
    lower_bound: lowest value of the arbitrary domain
    upper_bound: largest value of the arbitrary domain
    normalize:  if == 1 -> must be a value on the normalized domain
[0,1], and output will be in arbitrary domain
                if != 1 -> can be on an arbitrary domain and the output
will be in a normalized domain [0,1]
    """
    L = float(upper_bound - lower_bound)
    if normalize != 1: # value transfromed from normalized -> arbitrary
domain
        x = lower_bound + L * val
    elif normalize == 1: # value transformed from arbitrary domain -> n
ormalized domain
        x = (val - lower_bound) / L
    return x


def Trap(alpha, func, lower_bound, upper_bound, n_sub):
    """Performs the trapezoidal rule for the approximation of a definit
e integral
    alpha: weighting vector
    func: function to be evaluated
    lower_bound: function evaluated at lower boundary
    upper_bound: function evaluated at upper boundary
    n_sub: number of sub domains - MUST BE INTEGER
    """
    eta = linspace(0, 1, int(n_sub) + 1) # normalized subdomains [0,1]
    x = Domain_Trans(eta, lower_bound, upper_bound, 0) # global subdoma
ins [lower bound, upper bound]
    h = (upper_bound - lower_bound)*1.0 / n_sub # length of subdomain

    I_incv = np.zeros(int(n_sub)) # increment value for each subdomain

    for i in xrange(int(n_sub)):
        f_low = f(x[i])
        f_up = f(x[i+1])
        I_incv[i] = (alpha * f_up + (1 - alpha) * f_low) * h

    I_tot = np.sum(I_incv) # scalar total
    int_points = np.arange(lower_bound + h/2.0, upper_bound, h) # locat
ions on global domain where integration occured

    return I_tot, I_incv, int_points, h
```

Domonstrate that Trap works for alpha = 0.5 and 0.25

In [6]:
```
a = [0, 0.5, 1.0]
n_sub = 15
I_incv = np.zeros((n_sub, len(a)))

for i in xrange(len(a)):
    I_f, I_incv[:,i], int_points, h = Trap(a[i], f, x_min, x_max, n_su
b)
    display(I_f)
```

```
48.618454860218606

51.770447823255481

54.922440786292356
```

In [7]:
```
f_02, (ax1) = plt.subplots(1,1, sharex = True, sharey = True, figsize=
(12,8))
f_name = "02_Trap_Alpha.pdf"

# plot
ax1.plot(x,f_x,lw = 2, label = ''r'$f(x)=sin(x)+x$')
ax1.plot(int_points, I_incv[:,0]/h, marker = 'o', label = ''r'Trapezoid
al $\alpha = 0$')
ax1.plot(int_points, I_incv[:,1]/h, marker = 'o', label = ''r'Trapezoid
al $\alpha = 0.5$')
ax1.plot(int_points, I_incv[:,2]/h, marker = 'o', label = ''r'Trapezoid
al $\alpha = 1$')

ax1.grid(b = True, which = 'minor')
ax1.grid(b = True, which = 'major')
# plt.tight_layout()
ax1.legend(loc=0, fontsize = 15, framealpha = 1, fancybox = True)
ax1.set_xlabel(''r'x', fontsize = 25)
ax1.tick_params(axis = 'x', labelsize = 15)
ax1.tick_params(axis = 'y', labelsize = 15)

# ax.set_ylim(4, 9)
# # ax.set_xscale('log')
# # ax.set_yscale('log')
# plt.rcParams['font.size']=20


ax1.set_ylabel(''r'Functions of x', fontsize = 25)

ax1.set_title(''r'Integration Point Multiplied by Subdomain Width
$(h)$, $n_{subdomain}=15$', fontsize=20, y = 1.01)


path = "/Users/Lampe/Documents/UNM_Courses/ME-500/HW01"
out_file = path + "/" + f_name
f_02.savefig(out_file)
```



Prob 3:

Alpha = 0

In [8]:
```
a_0 = 0.
n_sub = np.array(([1, 2, 4, 8, 16, 32, 64]), dtype = np.int)
I_fv_0 = np.zeros((len(n_sub)))
h_v_0 = np.zeros(len(n_sub))

for i in xrange(len(n_sub)):
    I_fv_0[i], I_inc, int_points, h_v_0[i] = Trap(a_0, f, x_min, x_max,
n_sub[i])

# analytical solution = 51.839
print I_fv_0
Error_0 = np.abs(51.839 - I_fv_0)
```

```
[  0.      20.205   38.944   45.683   48.824   50.347   51.097]
```

Alpha = 0.5

In [9]:
```
a_05 = 0.5
n_sub = np.array(([1, 2, 4, 8, 16, 32, 64]), dtype = np.int)
I_fv_05 = np.zeros((len(n_sub)))
h_v_05 = np.zeros(len(n_sub))

for i in xrange(len(n_sub)):
    I_fv_05[i], I_inc, int_points, h_v_05[i] = Trap(a_05, f, x_min, x_m
ax, n_sub[i])

# analytical solution = 51.839
print I_fv_05
Error_05 = np.abs(51.839 - I_fv_05)
```

```
[ 47.28   43.845  50.764  51.593  51.779  51.824  51.835]
```

Alpha = 1

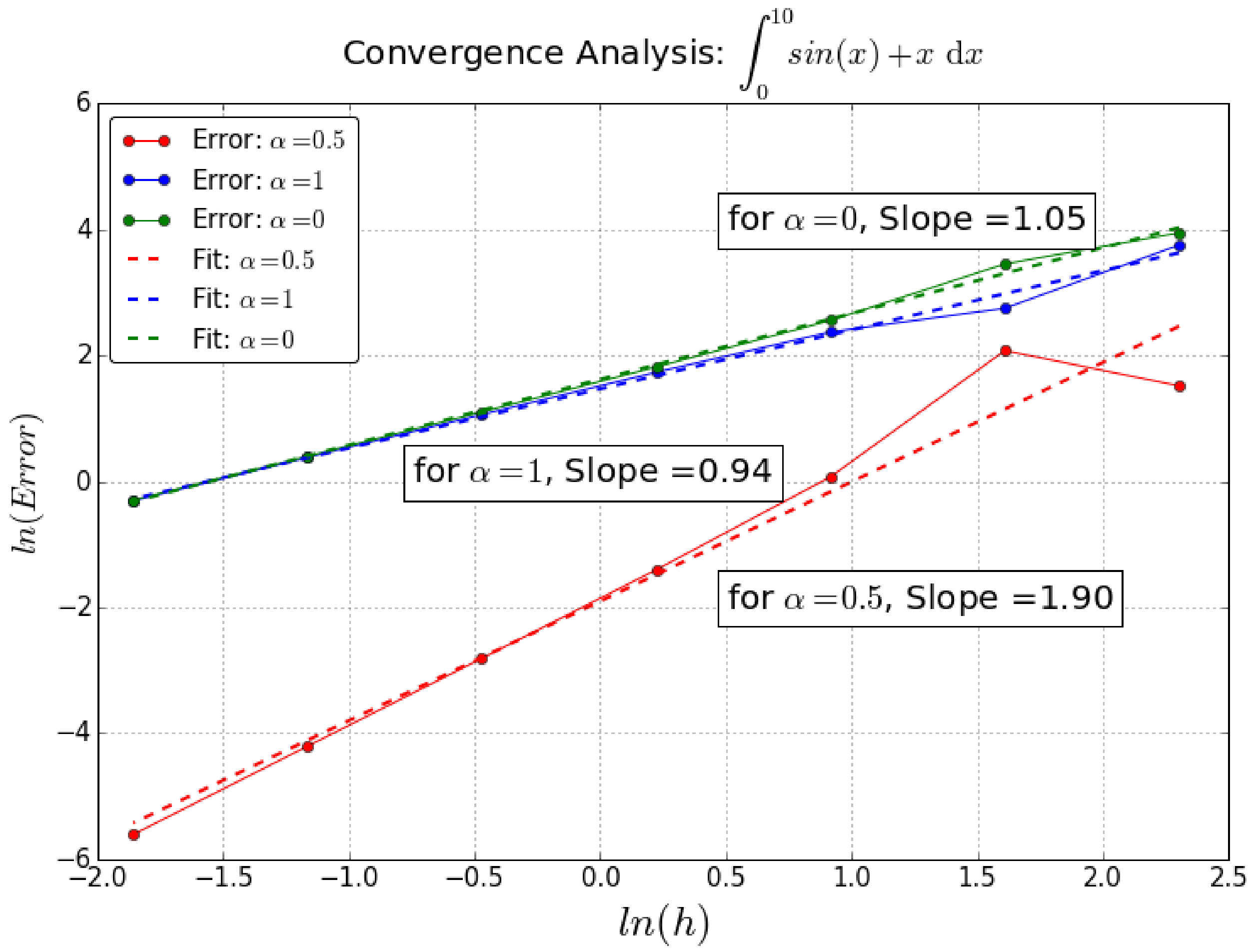
In [10]:
```
a_1 = 1.0
n_sub = np.array(([1, 2, 4, 8, 16, 32, 64]), dtype = np.int)
I_fv_1 = np.zeros((len(n_sub)))
h_v_1 = np.zeros(len(n_sub))

for i in xrange(len(n_sub)):
    I_fv_1[i], I_inc, int_points, h_v_1[i] = Trap(a_1, f, x_min, x_max,
n_sub[i])

# analytical solution = 51.839
print I_fv_1
Error_1 = np.abs(51.839 - I_fv_1)
```

```
[ 94.56   67.485  62.584  57.503  54.734  53.302  52.574]
```

Plot Results and Calculate the Rate of Convergence (slope)

In [11]:
```
f_03, (ax1) = plt.subplots(1,1, sharex = True, sharey = True, figsize=
(12,8))
f_name = "03_Trap_Error.pdf"

# plot
ax1.plot(np.log(h_v_05), np.log(Error_05), marker = 'o', color = 'red',
lw = 1, label = ''r'Error: $\alpha = 0.5$')
ax1.plot(np.log(h_v_1), np.log(Error_1), marker = 'o', lw = 1, label =
''r'Error: $\alpha = 1$')
ax1.plot(np.log(h_v_0), np.log(Error_0), marker = 'o', lw = 1, label =
''r'Error: $\alpha = 0$')

# fit to data
m_05, b_05 = np.polyfit(np.log(h_v_05), np.log(Error_05), 1)
m_1, b_1 = np.polyfit(np.log(h_v_1), np.log(Error_1), 1)
```

```
m_0, b_0 = np.polyfit(np.log(h_v_0), np.log(Error_0), 1)

# plot fits
ax1.plot(np.log(h_v_05), m_05 * np.log(h_v_05) + b_05, '--r',lw = 2, la
bel = ''r'Fit: $\alpha = 0.5$')
ax1.plot(np.log(h_v_1), m_1 * np.log(h_v_1) + b_1, '--b', lw = 2, label
= ''r'Fit: $\alpha = 1$')
ax1.plot(np.log(h_v_0), m_0 * np.log(h_v_0) + b_0, '--g', lw = 2, label
= ''r'Fit: $\alpha = 0$')

# annotate plots with text boxes
lbl_05 = ''r'for $\alpha = 0.5$, Slope ={:.2f}'.format(m_05)
lbl_1 = ''r'for $\alpha = 1$, Slope ={:.2f}'.format(m_1)
lbl_0 = ''r'for $\alpha = 0$, Slope ={:.2f}'.format(m_0)

ax1.text(0.5, -2.0, lbl_05, bbox={'facecolor':'white', 'pad':10}, fonts
ize = 20)
ax1.text(-.75, 0, lbl_1, bbox={'facecolor':'white', 'pad':10}, fontsize
= 20)
ax1.text(.5, 4, lbl_0, bbox={'facecolor':'white', 'pad':10}, fontsize =
20)

# format axis
ax1.grid(b = True, which = 'minor')
ax1.grid(b = True, which = 'major')
# plt.tight_layout()
ax1.legend(loc=0, fontsize = 15, framealpha = 1, fancybox = True)
ax1.tick_params(axis = 'x', labelsize = 15)
ax1.tick_params(axis = 'y', labelsize = 15)

# ax.set_ylim(4, 9)
# ax1.set_xscale('log')
# ax1.set_yscale('log')
# plt.rcParams['font.size']=20

#label axis and chart title
ax1.set_ylabel(''r'$ln(Error)$', fontsize = 20)
ax1.set_xlabel(''r'$ln(h)$', fontsize = 25)
ax1.set_title(''r'Convergence Analysis: $\int_0^{10} \!sin(x)+x \, \mat
hrm{d}x$', fontsize = 20, y=1.04, x=0.5)

# save to file
path = "/Users/Lampe/Documents/UNM_Courses/ME-500/HW01"
out_file = path + "/" + f_name
f_03.savefig(out_file)
# plt.show()
```



Prob 4:

Taylor series expansion of f(x) = [sin(x) + x] to 3 - 6 order polynomials and the subsequent integration

In [151]:
```
from sympy import *

x_1 = symbols('x')

a = 5.0
A = sin(a) + a
B = (cos(a) + 1)*(x_1-a)
C = -sin(a)*(x_1-a)**2 / factorial(2)
D = -cos(a)*(x_1-a)**3 / factorial(3)
E = sin(a)*(x_1-a)**4 / factorial(4)
F = cos(a)*(x_1-a)**5 / factorial(5)
G = -sin(a)*(x_1-a)**6 / factorial(6)
final_6 = A + B + C + D + E + F + G
final_5 = A + B + C + D + E + F
final_4 = A + B + C + D + E
final_3 = A + B + C + D

print integrate(final_6, (x_1, 0, 10))
print integrate(final_5, (x_1, 0, 10))
print integrate(final_4, (x_1, 0, 10))
print integrate(final_3, (x_1, 0, 10))
```

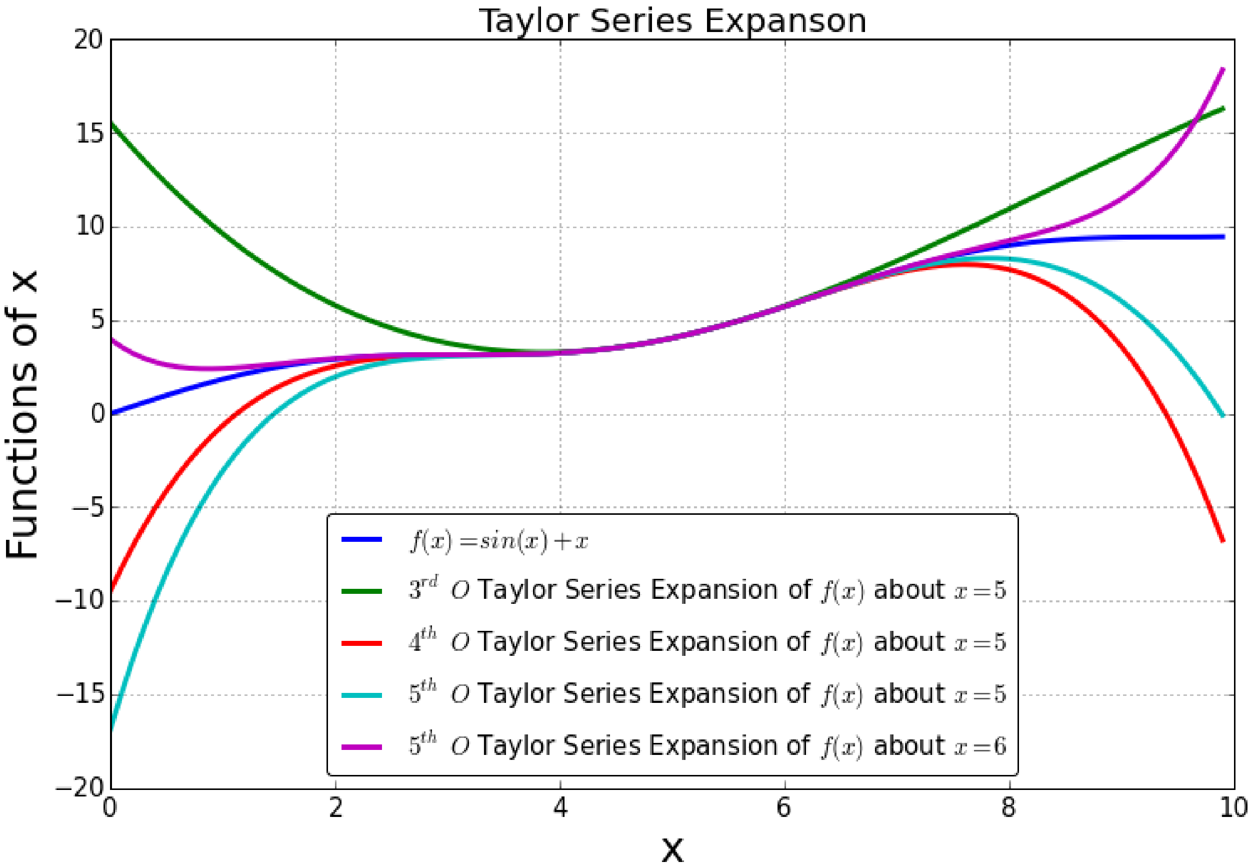
```
60.1505178677164
30.4219627256281
30.4219627256278
80.3659353643327
```

In [191]:
```
def TSE_a(x, **kwargs):
    """Taylor Series Expansion (TSE) of a function (f(x) = sin(x) + x)
about 5
    """
    poly_order = kwargs.get('poly_order', None)


    # TSEs of different orders
    if poly_order == 3:
        TSE = -0.0472770309100881*x**3 + 1.18861760099197*x**2 - 7.0567
3650614294*x + 15.5189470951249
    elif poly_order == 4:
        TSE = -0.0399551781105571*x**4 + 0.751826531302738*x**3 - 4.804
65911565407*x**2 + 12.9208525493389*x - 9.45303922422915
    elif poly_order == 5:
        TSE = 0.0023638515471667*x**5 - 0.0990514667493311*x**4 + 1.342
78941769049*x**3 - 7.75947354756246*x**2 + 20.3078886291125*x - 16.8400
753039966
    elif poly_order == 6:
        TSE = 0.00133183927003191*x**6 - 0.0375913265658449*x**5 + 0.40
0388259637272*x**4 - 1.98680875822193*x**3 + 4.72651961211457*x**2 -
4.66409769024094*x + 3.96991329546239
    else:
        print "Polynomial order defaults to 3"
        TSE = -0.0472770309100881*x**3 + 1.18861760099197*x**2 - 7.0567
3650614294*x + 15.5189470951249

    return TSE
```

In [192]:
```
# for plotting
TSE_3 = TSE_a(x, poly_order=3)
TSE_4 = TSE_a(x, poly_order=4)
TSE_5 = TSE_a(x, poly_order=5)
TSE_6 = TSE_a(x, poly_order=6)
TSE_test = TSE_a(x) # should default to evalution of 3rd order polynomi
al
```

```
Polynomial order defaults to 3
```

In [193]:
```
f_04, (ax1) = plt.subplots(1,1, sharex = True, sharey = True, figsize=
(12,8))
f_name = "04_GaussQuad.pdf"

# plot
ax1.plot(x,f_x,lw = 3, label = ''r'$f(x)=sin(x)+x$')
ax1.plot(x,TSE_3,lw = 3, label = ''r'$3^{rd} \, O$ Taylor Series Expans
ion of 'r'$f(x)$ about $x=5$')
ax1.plot(x,TSE_4,lw = 3, label = ''r'$4^{th} \, O$ Taylor Series Expans
ion of 'r'$f(x)$ about $x=5$')
ax1.plot(x,TSE_5,lw = 3, label = ''r'$5^{th} \, O$ Taylor Series Expans
ion of 'r'$f(x)$ about $x=5$')
ax1.plot(x,TSE_6,lw = 3, label = ''r'$5^{th} \, O$ Taylor Series Expans
ion of 'r'$f(x)$ about $x=6$')


ax1.grid(b = True, which = 'minor')
ax1.grid(b = True, which = 'major')
# plt.tight_layout()
ax1.legend(loc=0, fontsize = 15, framealpha = 1, fancybox = True)
ax1.set_xlabel(''r'x', fontsize = 25)
ax1.tick_params(axis = 'x', labelsize = 15)
ax1.tick_params(axis = 'y', labelsize = 15)

# ax.set_ylim(4, 9)
# # ax.set_xscale('log')
# # ax.set_yscale('log')
# plt.rcParams['font.size']=20

ax1.set_ylabel(''r'Functions of x', fontsize = 25)
ax1.set_title(''r'Taylor Series Expanson', fontsize = 20)

path = "/Users/Lampe/Documents/UNM_Courses/ME-500/HW01"
out_file = path + "/" + f_name
f_04.savefig(out_file)
```



Write Guass quadrature function

In [206]:
```
def GaussQuad_3(func, lower_bound, upper_bound, n_sub, **kwargs):
    """Performs third order guass quadrature - exact integral for 3rd O
polynomials
    func: function to be integrated
    lower_bound: lower_bound of integral
    upper_bound: upper_bound of integral
    n_sub: number of subdomains (integer value)

    """
    order = kwargs.get('poly_order', None)

    # integration points in normalized domain [0,1]
    ip_eta_1 = 0.5 - np.sqrt(3.0)/6.0
    ip_eta_2 = 0.5 + np.sqrt(3.0)/6.0

    # weight values
    w_1 = 0.5
    w_2 = 0.5

    # define the domain properties
    I_sub_x = np.zeros(n_sub)
    I_sub = np.zeros(n_sub)

    h = float(upper_bound - lower_bound) / n_sub # subdomain length
    LB = lower_bound
    UB = LB + h

    for i in xrange(int(n_sub)):
        ip_x_1 = Domain_Trans(ip_eta_1, LB, UB, 0)
        ip_x_2 = Domain_Trans(ip_eta_2, LB, UB, 0)

        I_sub_x[i] = w_1 * func(ip_x_1, poly_order = order) + w_2 * fun
c(ip_x_2, poly_order = order)
        I_sub[i] = I_sub_x[i] * h

        #update integration bounds
        LB = LB + h
        UB = UB + h

    I_num = np.sum(I_sub)

    return I_num
```

In [207]:
```
# test function
GaussQuad_3(TSE_a, 0, 10, 1)#, poly_order = 6)
```

```
Polynomial order defaults to 3
Polynomial order defaults to 3
```

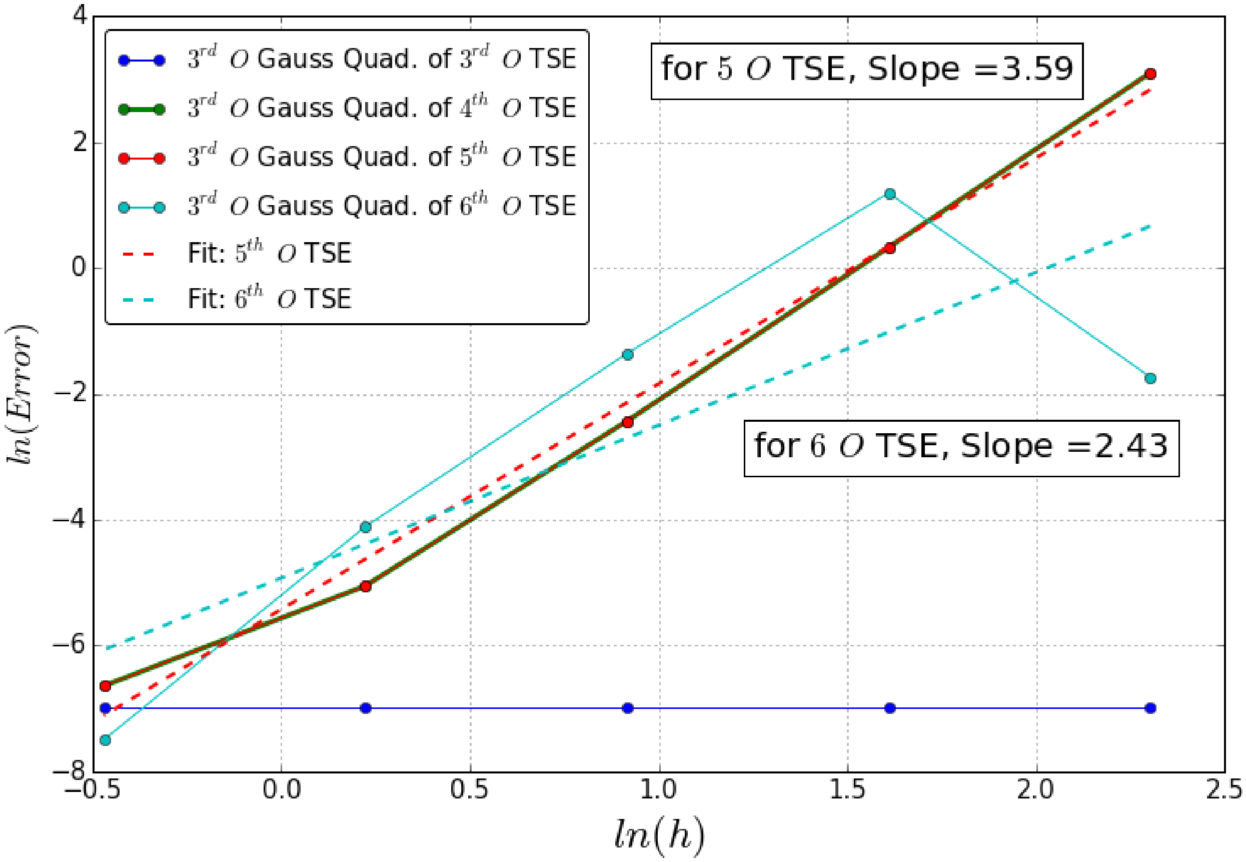
Out[207]:
```
80.365935366205093
```

Evaluation error using a different number of subdomains

In [222]:
```
n_domains = np.array(([1, 2, 4, 8, 16]), dtype = int)
LB = 0
UB = 10
h = float(UB - LB) / n_domains
gauss_int3 = np.zeros((len(n_domains),4))
gauss_error = np.zeros((len(n_domains),4))

for i in xrange(len(n_domains)):
#     gauss_int[i] = sp.integrate.fixed_quad(TSE_a, 0.0, 10.0, args=
([4]), n = gauss_order[i])[0]
    gauss_int3[i,0] = GaussQuad_3(TSE_a, LB, UB, n_sub=n_domains[i], po
ly_order = 3)
    gauss_int3[i,1] = GaussQuad_3(TSE_a, LB, UB, n_sub=n_domains[i], po
ly_order = 4)
    gauss_int3[i,2] = GaussQuad_3(TSE_a, LB, UB, n_sub=n_domains[i], po
ly_order = 5)
    gauss_int3[i,3] = GaussQuad_3(TSE_a, LB, UB, n_sub=n_domains[i], po
ly_order = 6)

    gauss_error[i,0] = np.abs(80.365 - gauss_int3[i,0])
    gauss_error[i,1] = np.abs(30.421 - gauss_int3[i,1])
    gauss_error[i,2] = np.abs(30.421 - gauss_int3[i,2])
    gauss_error[i,3] = np.abs(60.150 - gauss_int3[i,3])


# sp.integrate.fixed_quad(TSE_a, 0.0, 10.0, args=([3]), n = 3)
```

In [240]:
```
f_05, (ax1) = plt.subplots(1,1, sharex = True, sharey = True, figsize=
(12,8))
f_name = "05_Gauss_Error.pdf"

# plot
ax1.plot(np.log(h), np.log(gauss_error[:,0]), marker='o', lw=1, labe
l=''r'$3^{rd}\,O$ Gauss Quad. of $3^{rd}\,O$ TSE')
ax1.plot(np.log(h), np.log(gauss_error[:,1]), marker='o', lw=3, labe
l=''r'$3^{rd}\,O$ Gauss Quad. of $4^{th}\,O$ TSE')
ax1.plot(np.log(h), np.log(gauss_error[:,2]), marker='o', lw=1, labe
l=''r'$3^{rd}\,O$ Gauss Quad. of $5^{th}\,O$ TSE')
ax1.plot(np.log(h), np.log(gauss_error[:,3]), marker='o', lw=1, labe
l=''r'$3^{rd}\,O$ Gauss Quad. of $6^{th}\,O$ TSE')

# # fit to data
m_5, b_5 = np.polyfit(np.log(h), np.log(gauss_error[:,2]), 1)
m_6, b_6 = np.polyfit(np.log(h), np.log(gauss_error[:,3]), 1)

# # plot fits
ax1.plot(np.log(h), m_5 * np.log(h) + b_5, '--r',lw = 2, label = ''r'Fi
t: $5^{th}\,O$ TSE')
ax1.plot(np.log(h), m_6 * np.log(h) + b_6, '--c', lw = 2, label = ''r'F
```

```
it: $6^{th}\,O$ TSE')

# # annotate plots with text boxes
lbl_5 = ''r'for $5\,O$ TSE, Slope ={:.2f}'.format(m_5)
lbl_6 = ''r'for $6\,O$ TSE, Slope ={:.2f}'.format(m_6)

ax1.text(1, 3, lbl_5, bbox={'facecolor':'white', 'pad':10}, fontsize =
20)
ax1.text(1.25, -3, lbl_6, bbox={'facecolor':'white', 'pad':10}, fontsiz
e = 20)

# format axis
ax1.grid(b = True, which = 'minor')
ax1.grid(b = True, which = 'major')
# plt.tight_layout()
ax1.legend(loc=0, fontsize = 15, framealpha = 1, fancybox = True)
ax1.tick_params(axis = 'x', labelsize = 15)
ax1.tick_params(axis = 'y', labelsize = 15)
# ax1.set_xticks([1,2,3,4,5])
# ax.set_ylim(4, 9)
# ax1.set_xscale('log')
# ax1.set_yscale('log')
# plt.rcParams['font.size']=20

#label axis and chart title
ax1.set_ylabel(''r'$ln(Error)$', fontsize = 20)
ax1.set_xlabel(''r'$ln(h)$', fontsize = 25)
# ax1.set_title(''r'Convergence Analysis: $\int_0^{10} \!TSE(sin(x)+x)
\, \mathrm{d}x$', fontsize = 20, y=1.03, x=0.5)

# save to file
path = "/Users/Lampe/Documents/UNM_Courses/ME-500/HW01"
out_file = path + "/" + f_name
f_05.savefig(out_file)
# plt.show()
```



for 5 O TSE, Slope =3.59
for 6 O TSE, Slope =2.43
3^{rd} O Gauss Quad. of 3^{rd} O TSE
3^{rd} O Gauss Quad. of 4^{th} O TSE
3^{rd} O Gauss Quad. of 5^{th} O TSE
3^{rd} O Gauss Quad. of 6^{th} O TSE
Fit: 5^{th} O TSE
Fit: 6^{th} O TSE
$ln(Error)$
$ln(h)$
4
2
0
−2
−4
−6
−8
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


```
In [ ]:
```