In [ ]:
```
## CSUMS Examples Spring 2012
## A. O. Hausknecht
```

In [1]:
```
## Basic Python
print "10**50 =",10**50
print "sin(pi/2 =",sin(pi/2)
print "Integer division: 1/3 =",1/3,", Real Division: 1.0/3 =",1.0/3
print
```

```
10**50 = 100000000000000000000000000000000000000000000000000
sin(pi/2 = 1.0
Integer division: 1/3 = 0 , Real Division: 1.0/3 = 0.333333333333
```

In [2]:
```
## Variables
x = 3
y = 5
print 'x =',x, ', y =', y
print '%d + %d = %d'%(x,y,x+y)
print
```

```
x = 3 , y = 5
3 + 5 = 8
```

In [3]:
```
# Strings
x = 'Cat in the hat '
y = 'came back!'
print 'x =',x, 'y =',y
print "'%s' + '%s' = '%s'"%(x,y,x+y)
```

```
x = Cat in the hat  y = came back!
'Cat in the hat ' + 'came back!' = 'Cat in the hat came back!'
```

In [4]:
```
## Loops
print "%2s %10s %10s"%('n', 'n^2', 'n^3')
print "---------------------------------"
for i in range(10):
    print "%2g %10g %10g"%(i, i**2, i**3)
##
print
print "%2s %10s %10s"%('n', 'n^0.5', 'n^.25')
print "---------------------------------"
i = 0
while i < 10:
    print "%2g %10g %10g"%(i, i**0.5, i**0.25)
    i += 1
```

```
 n        n^2        n^3
---------------------------------
 0          0          0
```

```
 1           1           1
 2           4           8
 3           9          27
 4          16          64
 5          25         125
 6          36         216
 7          49         343
 8          64         512
 9          81         729

 n       n^0.5       n^.25
---------------------------------
 0           0           0
 1           1           1
 2     1.41421     1.18921
 3     1.73205     1.31607
 4           2     1.41421
 5     2.23607     1.49535
 6     2.44949     1.56508
 7     2.64575     1.62658
 8     2.82843     1.68179
 9           3     1.73205
```

In [5]:
```
## If-then-else
import random
for i in range(10):
    x = random.randrange(0,10)
    if x > 5:
        print "%g is too large!"%(x)
    elif 3 <= x and x <= 5:
        print "%g is within range!"%(x)
    else:
        print "%g is too small!"%(x)
```

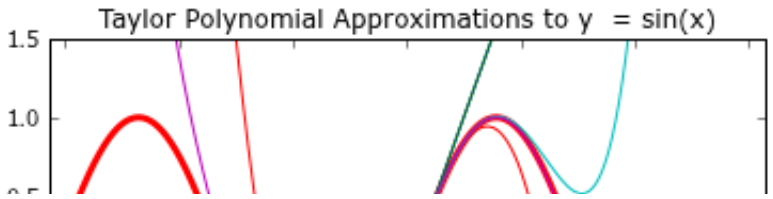
```
7 is too large!
1 is too small!
6 is too large!
3 is within range!
0 is too small!
6 is too large!
7 is too large!
8 is too large!
5 is within range!
8 is too large!
```

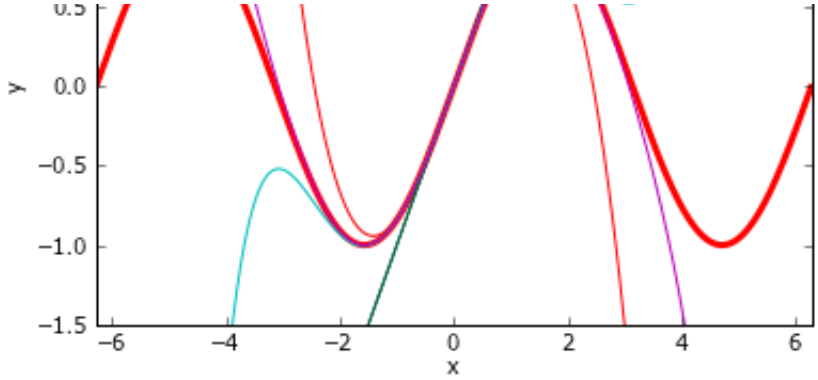
In [6]:
```
## Functions
def fact(n):
    if n <2:
        return 1
    else:
        return n*fact(n-1)

print "%d! = %d"%(50, fact(50))
```

```
def newtonRoot(a, n, tol = 1e-10):
    a = float(a)
    x1 = a/n+1; x2 = a/n
    count = 0
    while abs(x1 - x2)>tol:
        x1 = x2
        x2 = ((n-1)*x1 + a/x1**(n-1))/n
        count += 1
    return (x1, count)

an2 = newtonRoot(5,2); an3 = newtonRoot(5,3)
print "sqrt(%g) = %15.10g, n = %g; rad(3, %g) = %15.10g, n = %g"%(10, an2[0], an2
```

```
50! = 30414093201713378043612608166064768844377641568960512000000000000
sqrt(10) =     2.236067977, n = 5; rad(3, 10) =     1.709975947, n = 4
```

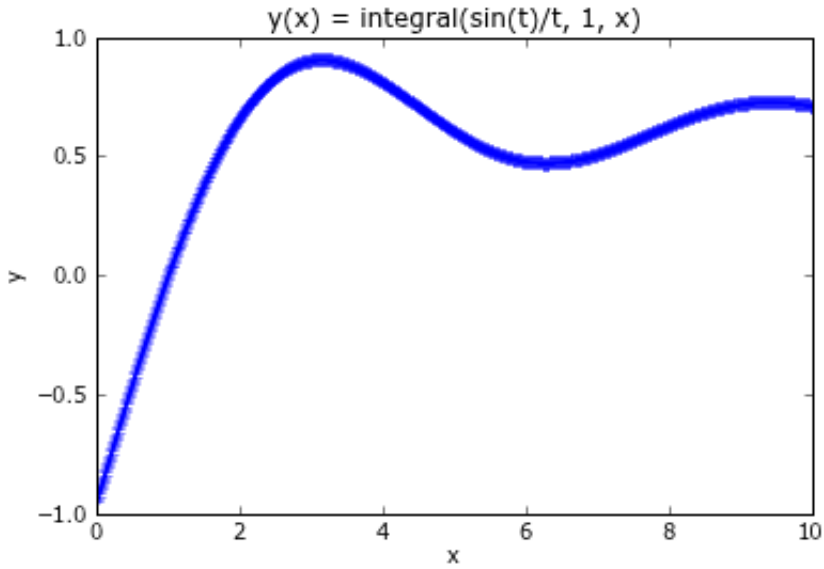
In [8]:
```
## Plot Taylor polynomial approximations to Sin(x)
## sin(x) = x - x^3/3! + x^5/5! - x^7/7! + ...
def taylorSin(x, n):
    s = x; xSqr = x**2; term = x;
    for i in range(1,n):
        twoI = 2*i
        term *= -xSqr/float((twoI+1)*twoI)
        s += term

    return s

## Generate the x-values
xValues = arange(-2*pi, 2*pi, .01)
n = len(xValues)
yValues = sin(xValues)
## Plot y = sin(x)
plot(xValues, sin(xValues), 'r', linewidth = 3)
axis([-2*pi, 2*pi, -1.5, 1.5])
## Now overlay plots of the Taylor polnomial approximations
for p in range(5):
    for i, x in zip(range(n), xValues):
        yValues[i] = taylorSin(x, p)
    plot(xValues, yValues)

title("Taylor Polynomial Approximations to y  = sin(x)")
xlabel('x'); ylabel('y');
```





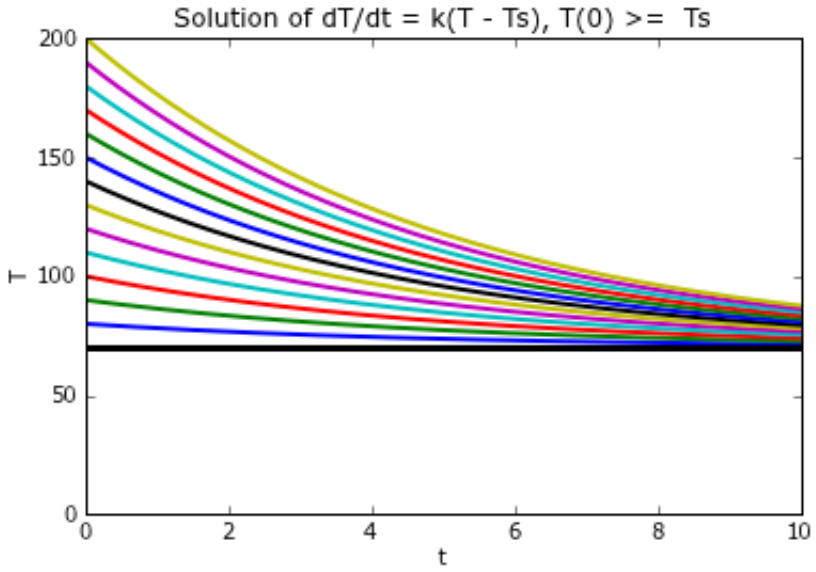
In [10]:
```
## Plot the function f(x) = integral(sin(t)/t, t, x)
##
from scipy.integrate import quad
##
xValues = arange(0, 10, .01)
yValues = zeros(len(xValues))
errors = zeros(len(xValues))
i=0
for x in xValues:
    yValues[i],errors[i] = quad(lambda t: sin(t)/t, 1, x)
    i += 1
print "Max Error: ", max(errors)
plot(xValues, yValues, '+-')

title("y(x) = integral(sin(t)/t, 1, x)")
xlabel('x'); ylabel('y');
```

```
Max Error:  1.79405622455e-14
```



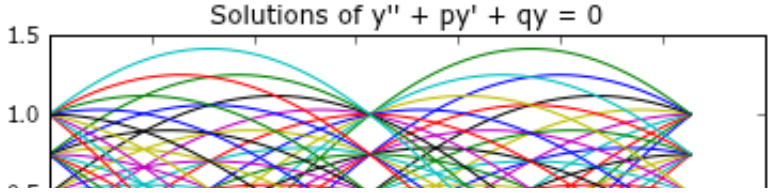
In [11]:
```
## Solution of dT/dt = k(T - Ts) = f(T)
##
from scipy.integrate import odeint
```

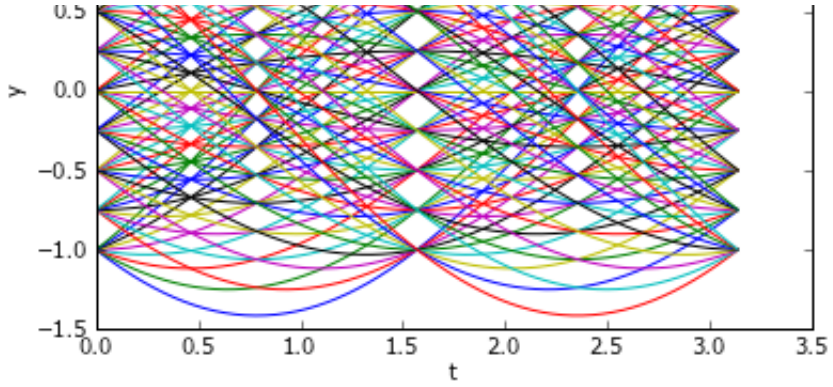
```
##
Ts = 70.0
k = -0.2
## Create a Pytho function that returns
##  f(T) as a 1 x 1 matrix.
def func(T, t):
    ## Note that T is a vector of length 1
    return [k*(T[0] - Ts)]

## Create a vector of t-values
t = arange(0, 10, .01)

## Using odeint, solve and plot the solutions
## of the ode for 70<= t0 <= 200
for T0 in arange(80, 210, 10):
    T = odeint(func, [T0], t)
    plot(t , T, linewidth = 2)

## Draw a horizonal line at at Ts
axhline(y=70, color = 'k', linewidth = 3)
## Set the axes limits
axis([0, 10, 0, 200])

title("Solution of dT/dt = k(T - Ts), T(0) >=  Ts")
xlabel('t'); ylabel('T');
```

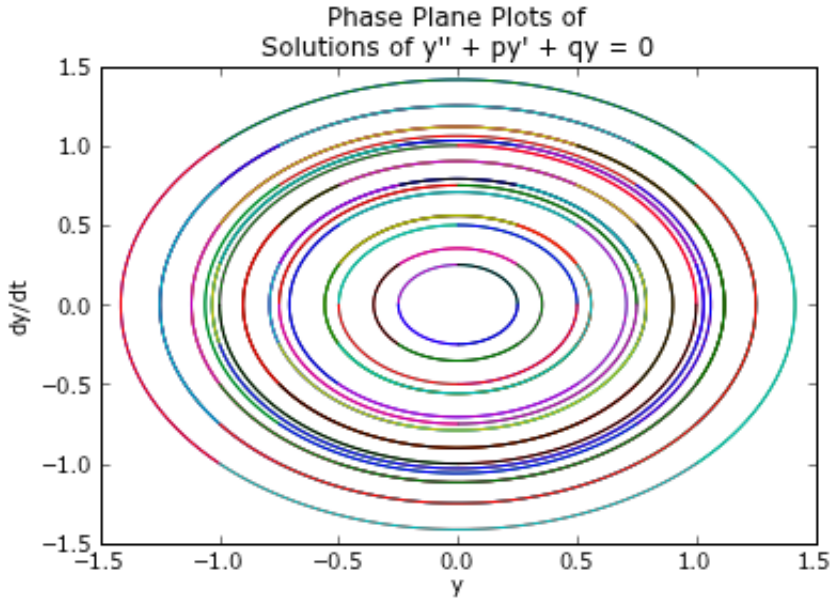


In [12]:
```
## Solution of y'' + p y' + q y = 0
## for various initial conditions.
##
## Step 1: Convert the 2nd-order linear ODE to
##         a first-order 2D system
##
## Let v = y', then  dv/dt = y''; hence,
##
```

```
##
##   dy/dt =  0*y + 1*v
##   dv/dt = -q*y - p*v
##
## Thus, in terms of matrices and vectors
##            [dy/dt]   [ 0  1] [y]
##      X' = [dv/dt] = [-q -p] [v]
##
from scipy.integrate import odeint
##
## Undamped
p = 0; q = 1;
##
## Over damped
## p = -1.0;  q = -2.0;
##
## Critally-damped
## p = 2.0; q = 1.0;
##
## Under-damped
## p = 2.0; q = 2.0;
##
def func(X, t):
    return [0*X[0]+1*X[1], -q*X[0]+-p*X[1]]

t = arange(0, pi, .01)

initialConds = []
for y0 in arange(-1, 1.25, 0.25):
    for dydt0 in arange(-1, 1.25, 0.25):
        initialConds.append([y0, dydt0])


figure(1); hold(True)
title("Solutions of y'' + py' + qy = 0")
xlabel('t'); ylabel('y');
figure(2); hold(True)
title("Phase Plane Plots of \nSolutions of y'' + py' + qy = 0")
xlabel('y'); ylabel('dy/dt');
for X0 in initialConds:
    X = odeint(func, X0, t)
    figure(1); plot(t, X[:,0])
    figure(2); plot(X[:,0], X[:,1])
```



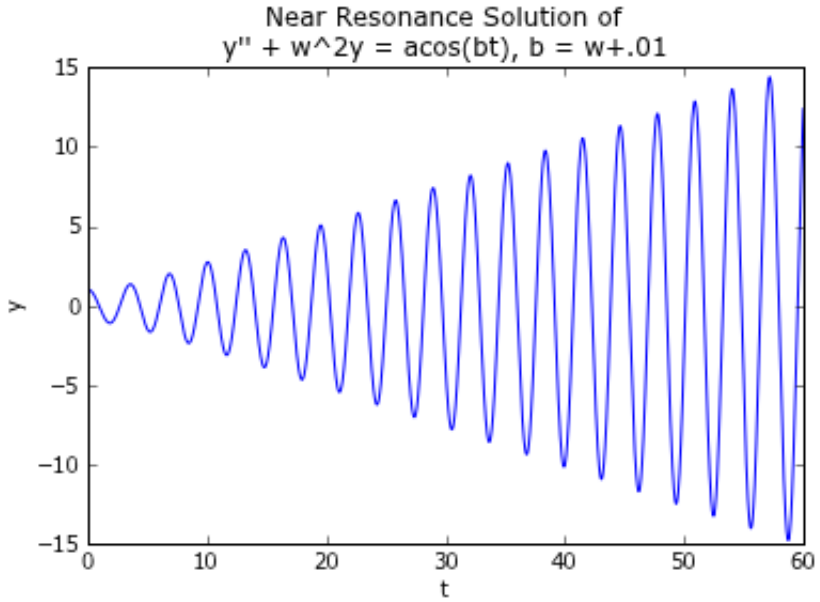


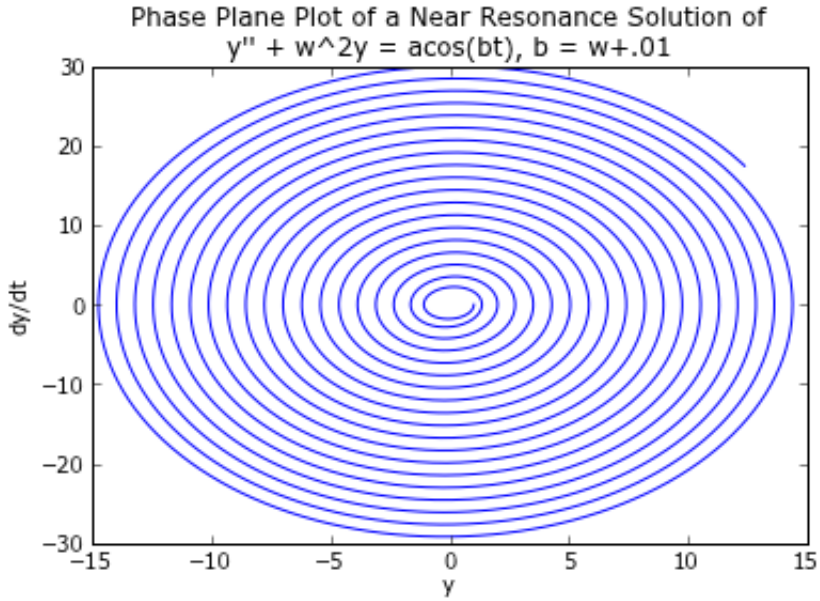


In [18]:
```
## Near Resonance
## Solution of y'' + w^2 y = a cos(b t)
##
from scipy.integrate import odeint
##
## In terms of matrices and vectors
##  [dy/dt]   [ 0  1] [y] + [  0     ]
##  [dv/dt] = [w^2 0] [v] * [a cos(b)]
w = 2.0; a = 1.0;
b = w + 0.01 ## Should be close to w;
def func(X, t):
    return [0*X[0]+1*X[1], -(w**2)*X[0] + 0*X[1] + a*cos(b*t)]

t = arange(0, 60, .01)

figure(1); hold(True)
title("Near Resonance Solution of\ny'' + w^2y = acos(bt), b = w+.01")
xlabel('t'); ylabel('y');
figure(2); hold(True)
title("Phase Plane Plot of a Near Resonance Solution of\ny'' + w^2y = acos(bt), b
xlabel('y'); ylabel('dy/dt');
X = odeint(func, [1, 0], t)
```

```
figure(1); plot(t, X[:,0])
figure(2); plot(X[:,0], X[:,1])
```

Out[18]: [<matplotlib.lines.Line2D at 0x7a15af0>]

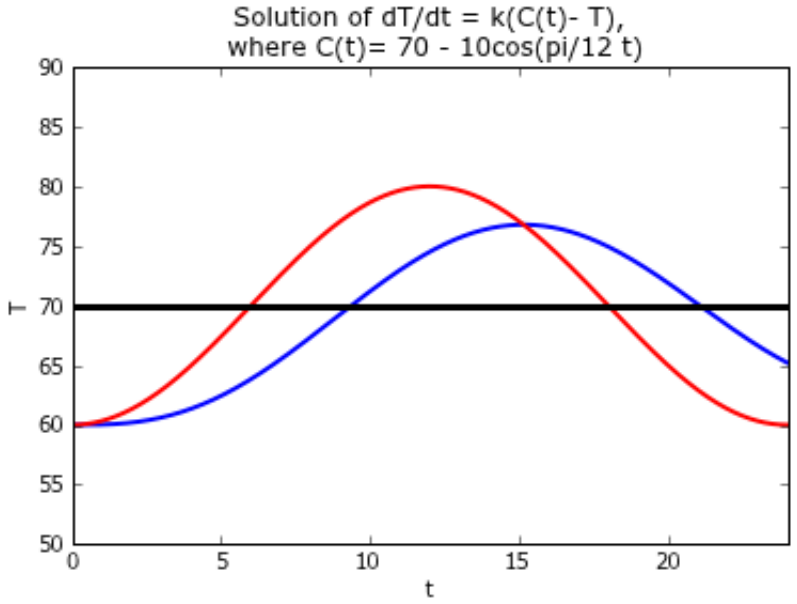




In [21]:
```
# Solution of dT/dt = k(C(t)- T) = f(T)
## T(t) = temperature of the inside of a barn
##        with no internal heating or cooling
##  where T(0) = 60 degrees
##  C(t) = 70 - 10*cos(pi/12*t), 0 <= t <= 24
##       = temperature outside the barn
##  k = 0.25 = temperature coefficient
from scipy.integrate import odeint
```

```
k = 0.25
def C(t):
    return 70 - 10*cos(pi/12*t)
## Create a Pytho function that returns
##  f(T) as a 1 x 1 matrix.
def func(T, t):
    ## Note that T is a vector of length 1
    return [k*(C(t) - T[0])]

## Create a vector of t-values
t = arange(0, 24, .01)

T = odeint(func, 60, t)
plot(t , T, linewidth = 2)
plot(t , C(t), 'r', linewidth = 2)
title("Solution of dT/dt = k(C(t)- T), \n where C(t)= 70 - 10cos(pi/12 t)")
xlabel('t'); ylabel('T');
## Draw a horizonal line at at Ts
axhline(y=70, color = 'k', linewidth = 3)
## Set the axes limits
axis([0, 24, 50, 90])
```

Out[21]: [0, 24, 50, 90]



In [ ]: