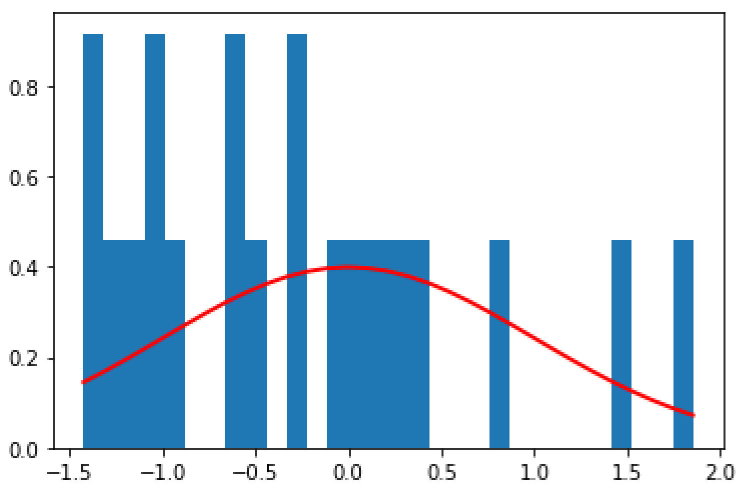
(a) Generate 20 data pairs (X, Y) using y = sin(2*pi*X) + N

In [17]:

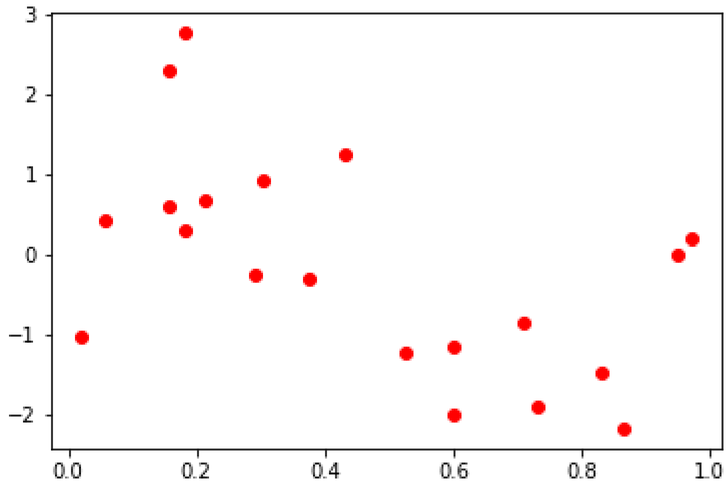
```
import numpy as np
import random
random.seed(42)
np.random.seed(42)
X= np.random.uniform(0,1,20)
print (X)
mu,sigma=0,1
N=np.random.normal(mu, sigma, 20)
print(N)
import matplotlib.pyplot as plt
count, bins, ignored = plt.hist(N, 30, density=True)
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
               np.exp( - (bins - mu)**2 / (2 * sigma**2) ),
         linewidth=2, color='r')
plt.show()
```

```
[0.37454012 0.95071431 0.73199394 0.59865848 0.15601864 0.15599452
 0.05808361 0.86617615 0.60111501 0.70807258 0.02058449 0.96990985
 0.83244264 0.21233911 0.18182497 0.18340451 0.30424224 0.52475643
 0.43194502 0.29122914]
[-1.01283112  0.31424733 -0.90802408 -1.4123037   1.46564877 -0.2257763
  0.0675282  -1.42474819 -0.54438272  0.11092259 -1.15099358  0.37569802
 -0.60063869 -0.29169375 -0.60170661  1.85227818 -0.01349722 -1.05771093
  0.82254491 -1.22084365]
```



In [18]:
```
import math
import matplotlib.pyplot as plt
Y=np.sin(2*np.pi*X)+N
plt.plot(X, Y, 'o', color='red')
```

Out[18]: [<matplotlib.lines.Line2D at 0x1e9e40c6d48>]



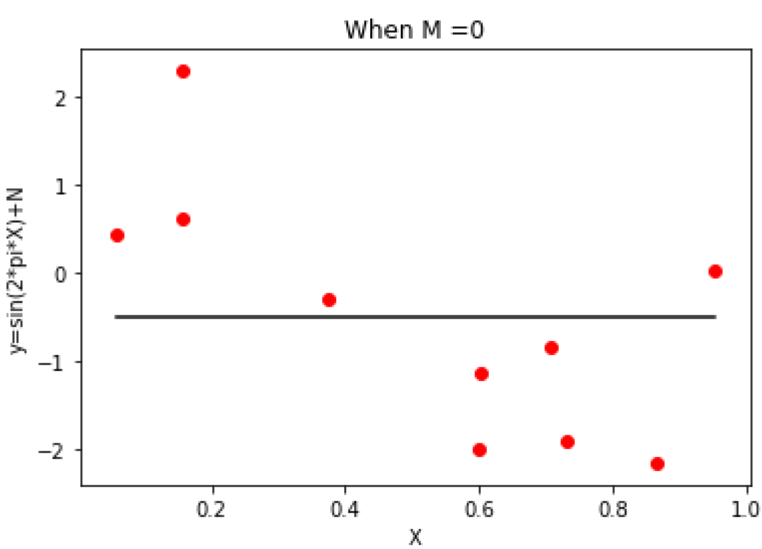
In [19]:
```
x_train=X[0:10]
print(x_train)
y_train=Y[0:10]
print(y_train)
x_test=X[10:20]
print(x_test)
y_test=Y[10:20]
print(y_test)
```

```
[0.37454012 0.95071431 0.73199394 0.59865848 0.15601864 0.15599452
 0.05808361 0.86617615 0.60111501 0.70807258]
[-0.3036841   0.00950189 -1.90163109 -1.99324896  2.29630988  0.60480043
  0.42443088 -2.16995176 -1.13782134 -0.8545779 ]
[0.02058449 0.96990985 0.83244264 0.21233911 0.18182497 0.18340451
 0.30424224 0.52475643 0.43194502 0.29122914]
[-1.02201767  0.18776035 -1.46944871  0.68043968  0.30794309  2.76600545
  0.92898565 -1.21263367  1.23723487 -0.25420997]
```

(b) Using root mean square error, find weights of polynomial regression for order is 0, 1, 3, 9 (d) Draw a chart of fit data

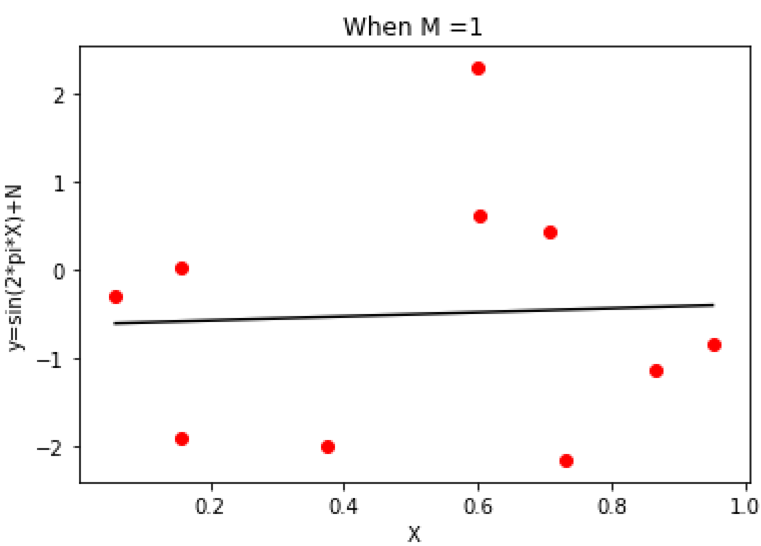
In [20]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import operator
from operator import itemgetter
from sklearn.metrics import mean_squared_error
from math import sqrt
x1=x_train[:,np.newaxis]
y1=y_train[:,np.newaxis]
poly = PolynomialFeatures(degree = 0)
X_poly = poly.fit_transform(x1)
poly.fit(X_poly, y1)
lin2 = LinearRegression()
lin2.fit(X_poly, y1)
y2=lin2.predict(X_poly)
rms = sqrt(mean_squared_error(y1, y2))
print('RMSE=',rms)
print('Cooefficient=',lin2.coef_)
plt.scatter(x1, y1, color='red')
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x1,y2), key=sort_axis)
x1, y2 = zip(*sorted_zip)
plt.plot(x1, y2, color='black')
plt.title('When M =0')
plt.xlabel('X')
plt.ylabel('y=sin(2*pi*X)+N')
plt.show()
```

```
RMSE= 1.3307221036666563
Cooefficient= [[0.]]
```



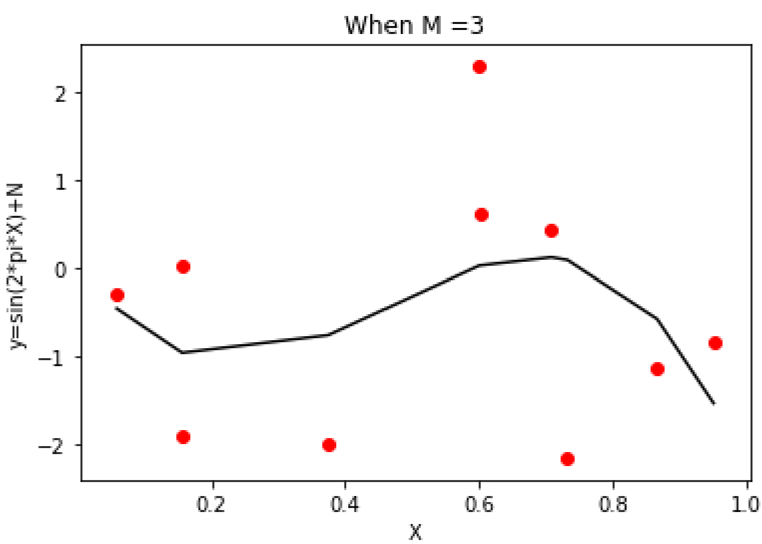
In [21]:
```
poly = PolynomialFeatures(degree = 1)
X_poly = poly.fit_transform(x1)
poly.fit(X_poly, y1)
lin2 = LinearRegression()
lin2.fit(X_poly, y1)
y2=lin2.predict(X_poly)
rms = sqrt(mean_squared_error(y1, y2))
print('RMSE=',rms)
print('Cooefficient=',lin2.coef_)
plt.scatter(x1, y1, color='red')
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x1,y2), key=sort_axis)
x1, y2 = zip(*sorted_zip)
plt.plot(x1, y2, color='black')
plt.title('When M =1')
plt.xlabel('X')
plt.ylabel('y=sin(2*pi*X)+N')
plt.show()
```

```
RMSE= 1.3289257344106773
Cooefficient= [[0.         0.23066742]]
```



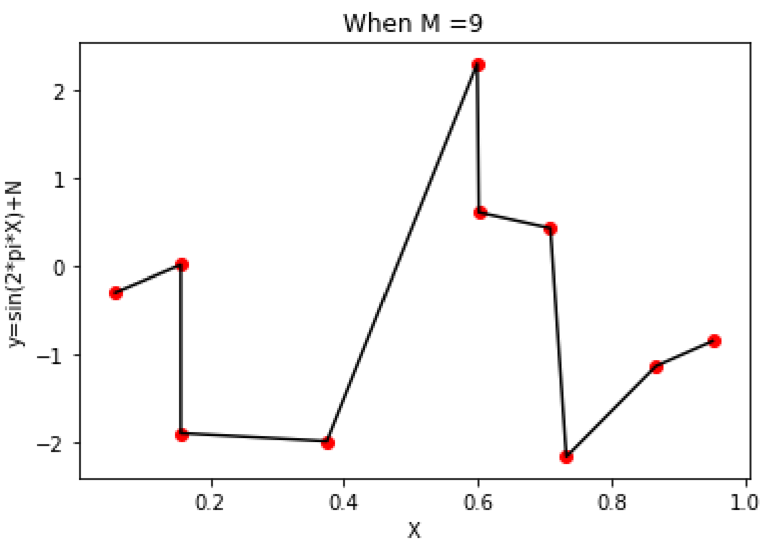
In [22]:
```
poly = PolynomialFeatures(degree = 3)
X_poly = poly.fit_transform(x1)
poly.fit(X_poly, y1)
lin2 = LinearRegression()
lin2.fit(X_poly, y1)
y2=lin2.predict(X_poly)
rms = sqrt(mean_squared_error(y1, y2))
print('RMSE=',rms)
print('Cooefficient=',lin2.coef_)
plt.scatter(x1, y1, color='red')
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x1,y2), key=sort_axis)
x1, y2 = zip(*sorted_zip)
plt.plot(x1, y2, color='black')
plt.title('When M =3')
plt.xlabel('X')
plt.ylabel('y=sin(2*pi*X)+N')
plt.show()
```

```
RMSE= 1.2185980456718613
Cooefficient= [[  0.         -11.43208875  33.62650285 -24.62218043]]
```



In [23]:
```
poly = PolynomialFeatures(degree = 9)
X_poly = poly.fit_transform(x1)
poly.fit(X_poly, y1)
lin2 = LinearRegression()
lin2.fit(X_poly, y1)
y2=lin2.predict(X_poly)
rms = sqrt(mean_squared_error(y1, y2))
print('RMSE=',rms)
print('Cooefficient=',lin2.coef_)
plt.scatter(x1, y1, color='red')
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x1,y2), key=sort_axis)
x1, y2 = zip(*sorted_zip)
plt.plot(x1, y2, color='black')
plt.title('When M =9')
plt.xlabel('X')
plt.ylabel('y=sin(2*pi*X)+N')
plt.show()
```

```
RMSE= 1.0861921666330498e-07
Cooefficient= [[ 0.00000000e+00  1.76480723e+06 -2.11436975e+07  1.25276252e+
08
  -4.26092448e+08  8.90250118e+08 -1.16540180e+09  9.33878997e+08
  -4.19476059e+08  8.09974015e+07]]
```



(c) Display weights in table

In [24]:
```
from prettytable import PrettyTable
table = PrettyTable()

table.field_names = ["coefficients", "M=0", "M=1", "M=3", "M=9"]

table.add_row(["w0",0., 0., 0.,                        0.])
table.add_row(["w1",'' ,0.23066742, -11.43208875, 1.76480723e+06])
table.add_row(["w2",'' , '',  33.62650285 ,     -2.11436975e+07])
table.add_row(["w3",'' , '', -24.62218043,          1.25276252e+08])
table.add_row(["w4",'' , '','',       -4.26092448e+08])
table.add_row(["w5",'' , '', '',          8.90250118e+08])
table.add_row(["w6",'' , '', '',          -1.16540180e+09])
table.add_row(["w7",'' , '', '',           9.33878997e+08])
table.add_row(["w8",'' , '', '',           -4.19476059e+08])
table.add_row(["w9",'' , '', '',           8.09974015e+07])


print(table)
```

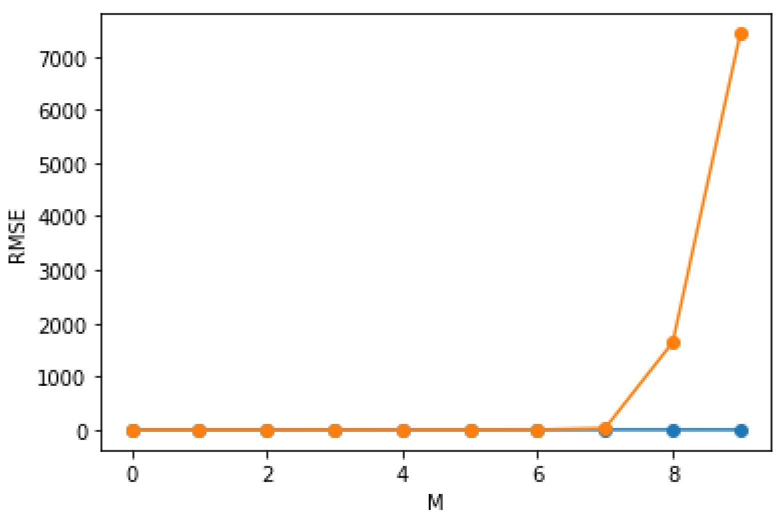
```
+--------------+-----+------------+--------------+---------------+
| coefficients | M=0 |    M=1     |     M=3      |      M=9      |
+--------------+-----+------------+--------------+---------------+
|      w0      | 0.0 |    0.0     |     0.0      |      0.0      |
|      w1      |     | 0.23066742 | -11.43208875 |   1764807.23  |
|      w2      |     |            | 33.62650285  |  -21143697.5  |
|      w3      |     |            | -24.62218043 |  125276252.0  |
|      w4      |     |            |              |  -426092448.0 |
|      w5      |     |            |              |  890250118.0  |
|      w6      |     |            |              | -1165401800.0 |
|      w7      |     |            |              |  933878997.0  |
|      w8      |     |            |              |  -419476059.0 |
|      w9      |     |            |              |   80997401.5  |
+--------------+-----+------------+--------------+---------------+
```

(e) Draw train error vs test error

In [25]:
```
xt=x_test[:,np.newaxis]
yt=y_test[:,np.newaxis]
train_error=list()
test_error=list()
degrees=[0,1,2,3,4,5,6,7,8,9]
for i in range(len(degrees)):
    poly = PolynomialFeatures(degree = i)
    X_poly = poly.fit_transform(x1)
    poly.fit(X_poly, y1)
    lin2 = LinearRegression()
    lin2.fit(X_poly, y1)
    y2=lin2.predict(X_poly)
    rms = np.sqrt(mean_squared_error(y1, y2))
    train_error=np.append(train_error,rms)
    rmse_test = np.sqrt(mean_squared_error(yt, lin2.predict(poly.fit_transform
(xt))))
    test_error=np.append(test_error, rmse_test)

print(train_error)
print(test_error)
plt.plot(degrees,train_error, marker='o')
plt.plot(degrees,test_error, marker='o')
plt.ylabel('RMSE')
plt.xlabel('M')
```

```
[1.33072210e+00 1.32892573e+00 1.27891838e+00 1.21859805e+00
 1.10805704e+00 9.61510895e-01 7.40133643e-01 6.05378458e-01
 3.21965094e-01 1.08619217e-07]
[1.41646242e+00 1.45036043e+00 1.43237805e+00 1.83459123e+00
 2.13504028e+00 2.10791197e+00 3.17830978e+00 2.90664081e+01
 1.64069715e+03 7.42911014e+03]
```

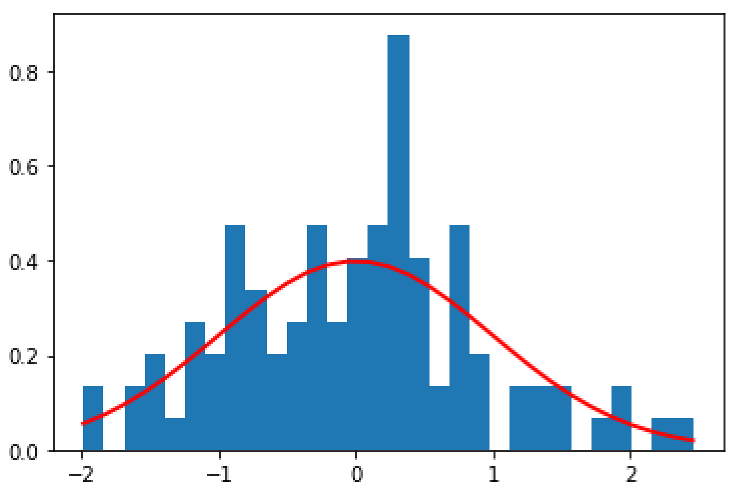
Out[25]: Text(0.5, 0, 'M')



(f) Now generate 100 more data and fit 9th order model and draw fit

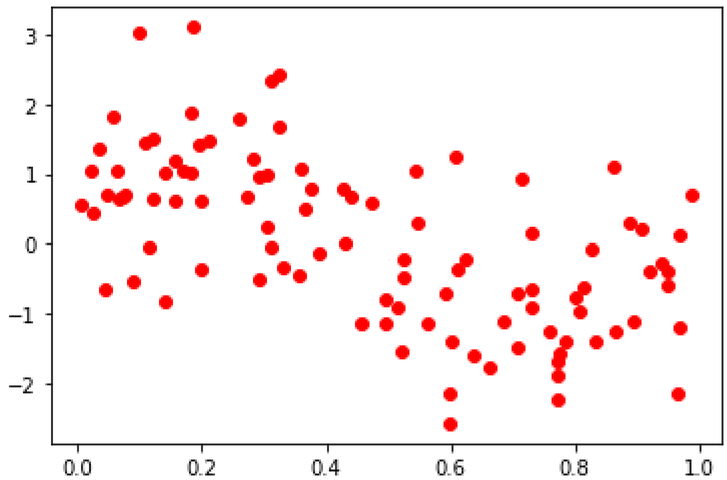
In [26]:
```
random.seed(42)
np.random.seed(42)
A= np.random.uniform(0,1,100)
print (A)
mu,sigma=0,1
B=np.random.normal(mu, sigma, 100)
print(B)
import matplotlib.pyplot as plt
count, bins, ignored = plt.hist(B, 30, density=True)
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
               np.exp( - (bins - mu)**2 / (2 * sigma**2) ),
         linewidth=2, color='r')
plt.show()
```

```
[0.37454012 0.95071431 0.73199394 0.59865848 0.15601864 0.15599452
 0.05808361 0.86617615 0.60111501 0.70807258 0.02058449 0.96990985
 0.83244264 0.21233911 0.18182497 0.18340451 0.30424224 0.52475643
 0.43194502 0.29122914 0.61185289 0.13949386 0.29214465 0.36636184
 0.45606998 0.78517596 0.19967378 0.51423444 0.59241457 0.04645041
 0.60754485 0.17052412 0.06505159 0.94888554 0.96563203 0.80839735
 0.30461377 0.09767211 0.68423303 0.44015249 0.12203823 0.49517691
 0.03438852 0.9093204  0.25877998 0.66252228 0.31171108 0.52006802
 0.54671028 0.18485446 0.96958463 0.77513282 0.93949894 0.89482735
 0.59789998 0.92187424 0.0884925  0.19598286 0.04522729 0.32533033
 0.38867729 0.27134903 0.82873751 0.35675333 0.28093451 0.54269608
 0.14092422 0.80219698 0.07455064 0.98688694 0.77224477 0.19871568
 0.00552212 0.81546143 0.70685734 0.72900717 0.77127035 0.07404465
 0.35846573 0.11586906 0.86310343 0.62329813 0.33089802 0.06355835
 0.31098232 0.32518332 0.72960618 0.63755747 0.88721274 0.47221493
 0.11959425 0.71324479 0.76078505 0.5612772  0.77096718 0.4937956
 0.52273283 0.42754102 0.02541913 0.10789143]
[ 0.08704707 -0.29900735  0.09176078 -1.98756891 -0.21967189  0.35711257
  1.47789404 -0.51827022 -0.8084936  -0.50175704  0.91540212  0.32875111
 -0.5297602   0.51326743  0.09707755  0.96864499 -0.70205309 -0.32766215
 -0.39210815 -1.46351495  0.29612028  0.26105527  0.00511346 -0.23458713
 -1.41537074 -0.42064532 -0.34271452 -0.80227727 -0.16128571  0.40405086
  1.8861859   0.17457781  0.25755039 -0.07444592 -1.91877122 -0.02651388
  0.06023021  2.46324211 -0.19236096  0.30154734 -0.03471177 -1.16867804
  1.14282281  0.75193303  0.79103195 -0.90938745  1.40279431 -1.40185106
  0.58685709  2.19045563 -0.99053633 -0.56629773  0.09965137 -0.50347565
 -1.55066343  0.06856297 -1.06230371  0.47359243 -0.91942423  1.54993441
 -0.78325329 -0.32206152  0.81351722 -1.23086432  0.22745993  1.30714275
 -1.60748323  0.18463386  0.25988279  0.78182287 -1.23695071 -1.32045661
  0.52194157  0.29698467  0.25049285  0.34644821 -0.68002472  0.2322537
  0.29307247 -0.71435142  1.86577451  0.47383292 -1.1913035   0.65655361
 -0.97468167  0.7870846   1.15859558 -0.82068232  0.96337613  0.41278093
  0.82206016  1.89679298 -0.24538812 -0.75373616 -0.88951443 -0.81581028
 -0.07710171  0.34115197  0.2766908   0.82718325]
```



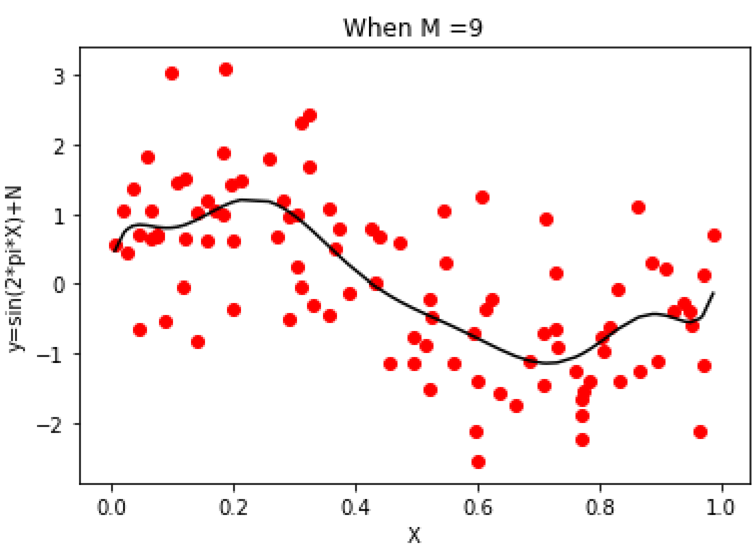
In [27]:
```
C=np.sin(2*np.pi*A)+B
plt.plot(A, C, 'o', color='red')
```

Out[27]: [<matplotlib.lines.Line2D at 0x1e9e3f7d708>]



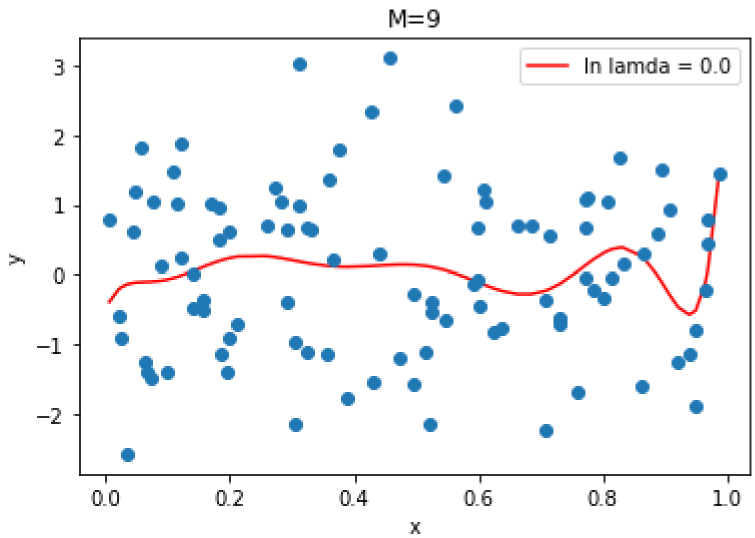
In [28]:
```
random.seed(42)
np.random.seed(42)
a1=A[:,np.newaxis]
c1=C[:,np.newaxis]
poly = PolynomialFeatures(degree = 9)
A_poly = poly.fit_transform(a1)
poly.fit(A_poly, c1)
lin2 = LinearRegression()
lin2.fit(A_poly, c1)
c2=lin2.predict(A_poly)
rms = sqrt(mean_squared_error(c1,c2))
print('RMSE=',rms)
print('Cooefficient=',lin2.coef_)
plt.scatter(a1, c1, color='red')
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(a1,c2), key=sort_axis)
a1, c2 = zip(*sorted_zip)
plt.plot(a1, c2, color='black')
plt.title('When M =9')
plt.xlabel('X')
plt.ylabel('y=sin(2*pi*X)+N')
plt.show()
```

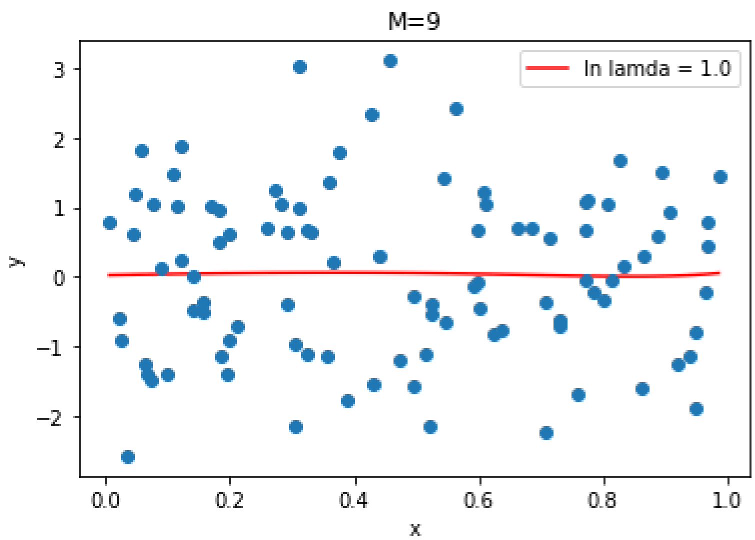
```
RMSE= 0.8673906703972919
Cooefficient= [[ 0.00000000e+00  3.24189288e+01 -7.13269208e+02  7.17506899e+
03
  -3.69590556e+04  1.06904740e+05 -1.81926071e+05  1.80992288e+05
  -9.73972757e+04  2.18912201e+04]]
```

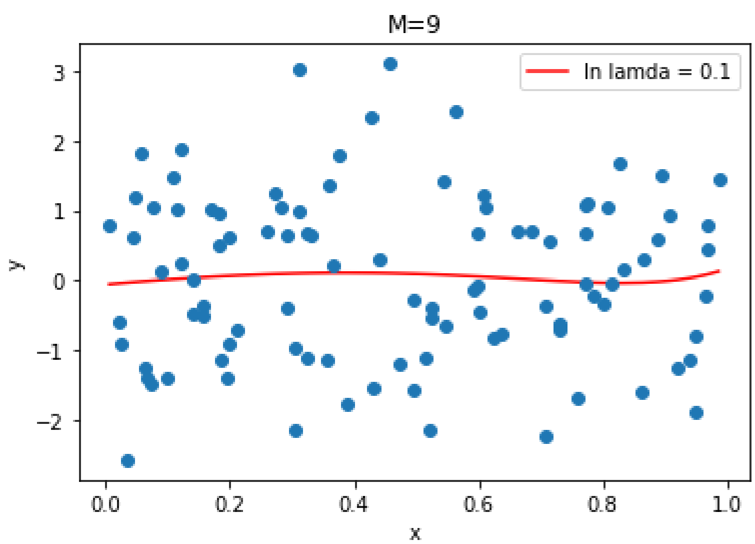


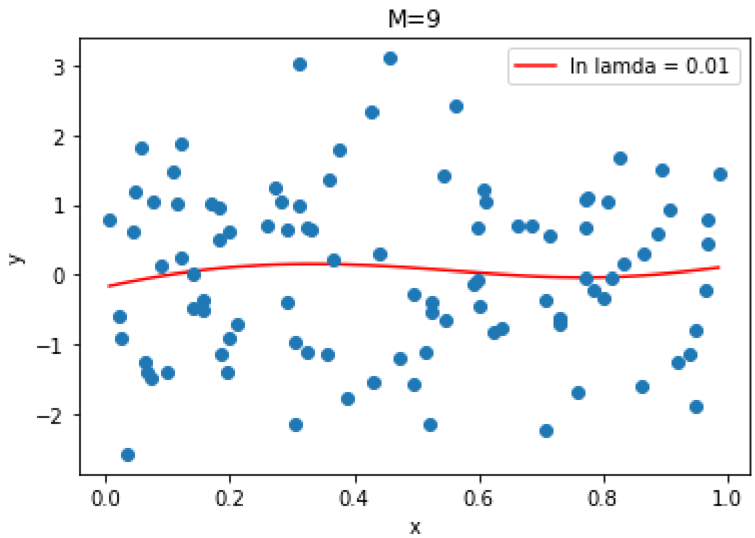
(g) Now we will regularize using the sum of weights (h) Draw chart for lambda is 1, 1/10, 1/100, 1/1000, 1/10000, 1/100000

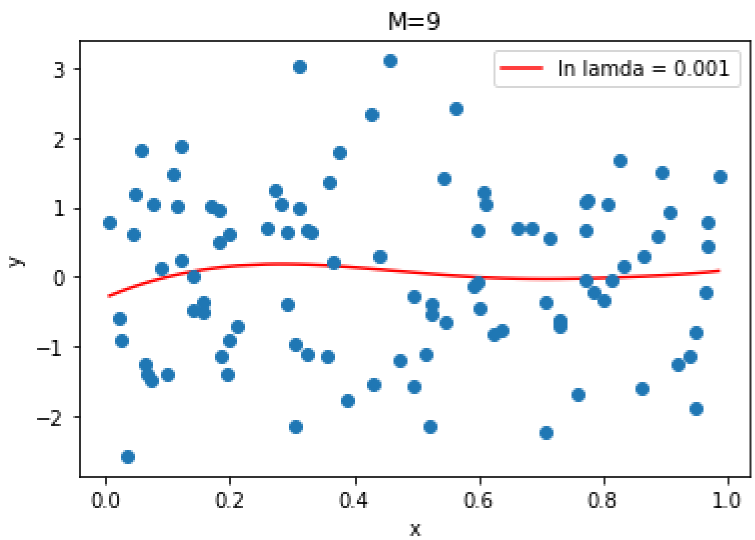
In [29]:
```
from sklearn.linear_model import Ridge
def plot_lam_graphs(degree,lambda_value):
    polynomial_features= PolynomialFeatures(degree)
    A_Poly = polynomial_features.fit_transform(a1)
    polynomial_features.fit(A_poly,c1)
    model = Ridge(alpha=lambda_value)
    fit = model.fit(A_Poly,c1)
    A_poly_plot = polynomial_features.fit_transform(a1)
    C_pred = model.predict(A_poly_plot)
    plt.plot( a1, C_pred, color='r',label="ln lamda = "+str(lambda_value))
    plt.plot(a1,c1,'o')
    plt.xlabel('x')#xdata ranging from 0 to 1
    plt.ylabel('y')#t is the target yalue i.e value of y
    plt.title("M="+str(degree))#M no. of coeffecients
    plt.legend()
    plt.show()

lambda_values = np.array([0,1, 0.1, 0.01, 0.001, 0.0001,0.00001])
for i in lambda_values:
    plot_lam_graphs(9,i)
```

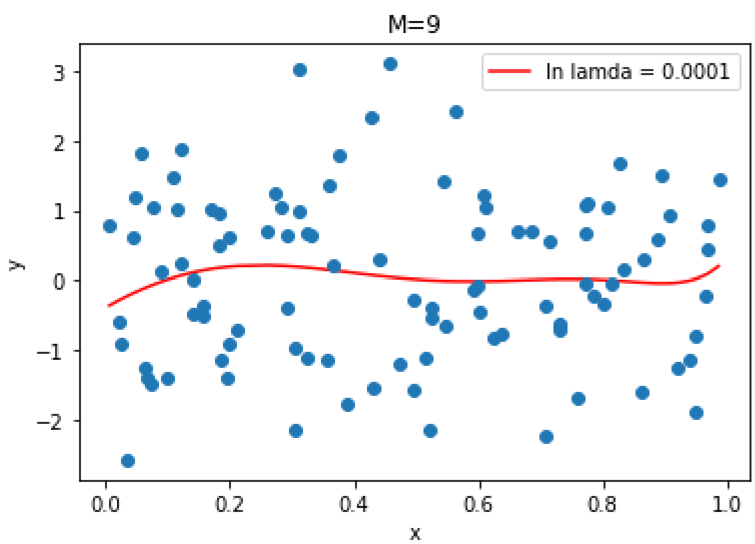

M=9
ln lamda = 0.0
y
x
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0

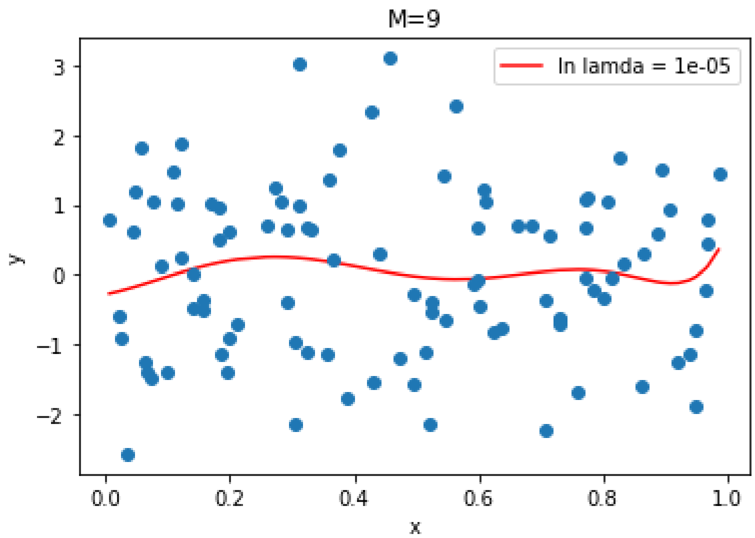

M=9
ln lamda = 1.0
y
x
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0


M=9
ln lamda = 0.1
y
x
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0


M=9
ln lamda = 0.01
y
x


M=9
ln lamda = 0.001
y
x


M=9
ln lamda = 0.0001
y
x



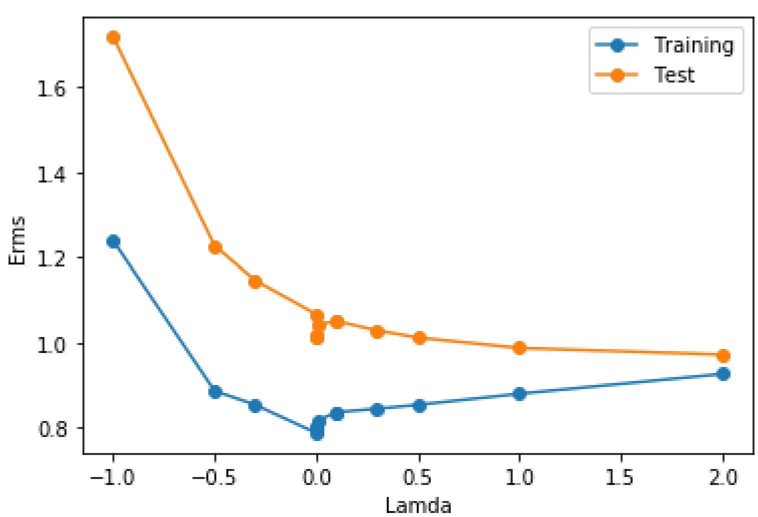
(i) Now draw test and train error according to lamda

In [30]:
```
a_train=A[0:50]
print(a_train)
c_train=C[0:50]
print(c_train)
a_test=A[50:100]
print(a_test)
c_test=C[50:100]
print(c_test)
a1=a_train[:,np.newaxis]
c1=c_train[:,np.newaxis]
at=a_test[:,np.newaxis]
ct=c_test[:,np.newaxis]
```

```
[0.37454012 0.95071431 0.73199394 0.59865848 0.15601864 0.15599452
 0.05808361 0.86617615 0.60111501 0.70807258 0.02058449 0.96990985
 0.83244264 0.21233911 0.18182497 0.18340451 0.30424224 0.52475643
 0.43194502 0.29122914 0.61185289 0.13949386 0.29214465 0.36636184
 0.45606998 0.78517596 0.19967378 0.51423444 0.59241457 0.04645041
 0.60754485 0.17052412 0.06505159 0.94888554 0.96563203 0.80839735
 0.30461377 0.09767211 0.68423303 0.44015249 0.12203823 0.49517691
 0.03438852 0.9093204  0.25877998 0.66252228 0.31171108 0.52006802
 0.54671028 0.18485446]
[ 0.79619409 -0.60375279 -0.90184624 -2.56851417  0.61098923  1.1876893
  1.83479672 -1.26347379 -1.40193222 -1.46725753  1.04437803  0.14081344
 -1.39857022  1.48540087  1.00672725  1.88237226  0.24042978 -0.48258489
  0.02258181 -0.49688126 -0.35023069  1.02953751  0.97025763  0.50983789
 -1.14284186 -1.39632031  0.60770661 -0.89159569 -0.70985981  0.69178162
  1.26072333  1.05246623  0.65499577 -0.39011493 -2.1330372  -0.95995008
  1.00193024  3.0391318  -1.10819107  0.66878086  0.65911456 -1.13837831
  1.35721495  0.21250582  1.78951068 -1.76210077  2.32855943 -1.5276083
  0.29756298  3.10784668]
[0.96958463 0.77513282 0.93949894 0.89482735 0.59789998 0.92187424
 0.0884925  0.19598286 0.04522729 0.32533033 0.38867729 0.27134903
 0.82873751 0.35675333 0.28093451 0.54269608 0.14092422 0.80219698
 0.07455064 0.98688694 0.77224477 0.19871568 0.00552212 0.81546143
 0.70685734 0.72900717 0.77127035 0.07404465 0.35846573 0.11586906
 0.86310343 0.62329813 0.33089802 0.06355835 0.31098232 0.32518332
 0.72960618 0.63755747 0.88721274 0.47221493 0.11959425 0.71324479
 0.76078505 0.5612772  0.77096718 0.4937956  0.52273283 0.42754102
 0.02541913 0.10789143]
[-1.18048063 -1.55385517 -0.27139852 -1.1172395  -2.12772299 -0.402838
 -0.53449822  1.41654712 -0.63906202  2.43999674 -0.13944778  0.66895521
 -0.06658335 -0.44750919  1.20863001  1.04208154 -0.83328143 -0.7620665
  0.71135583  0.69952425 -2.22719905 -0.37192468  0.55663109 -0.61961467
 -0.71299121 -0.64486535 -1.67110744  0.68088767  1.06969466 -0.04895348
  1.10783635 -0.22567235 -0.31772903  1.04537216 -0.04719498  1.67756761
  0.16679405 -1.58132275  0.31255669  0.58647426  1.50474659  0.92334127
 -1.24309298 -1.12931012 -1.88084917 -0.77683674 -0.21945111  0.78085969
  0.43572575  1.45434341]
```

In [31]:
```
training=[]
test=[]
lambda_values = np.array([-1,-0.5,-0.3,0,0.00001,0.0001,0.001,0.01,0.1,0.1,0.3
,0.5,1,2])

for i in lambda_values:
    poly_features = PolynomialFeatures(9)
    A_poly = poly_features.fit_transform(a1)
    poly_model = Ridge(alpha=i)
    poly_model.fit(A_poly, c1)
    A_poly_plot = poly_features.fit_transform(a1)
    C_predicted = poly_model.predict(A_poly_plot)
    rmse_train = sqrt(mean_squared_error(c1, C_predicted))
    training.append(rmse_train)
    rmse_test = sqrt(mean_squared_error(ct, poly_model.predict(poly_features.f
it_transform(at))))
    test.append(rmse_test)
print(lambda_values)
print(training)
print(test)
plt.plot(lambda_values,training,'o',linestyle='solid',label='Training')
plt.plot(lambda_values,test,'o',linestyle='solid',label='Test')
plt.xlabel('Lamda')
plt.ylabel('Erms')
plt.legend()
plt.show()
```

```
[-1.e+00 -5.e-01 -3.e-01  0.e+00  1.e-05  1.e-04  1.e-03  1.e-02  1.e-01
  1.e-01  3.e-01  5.e-01  1.e+00  2.e+00]
[1.2385893896410014, 0.8858072216517349, 0.8530988195501283, 0.78696768118439
8, 0.7978915038362975, 0.7983729646604112, 0.8001313381508149, 0.816393812335
3369, 0.8355310591001678, 0.8355310591001678, 0.8437498012252642, 0.852971155
1400033, 0.8789731427498115, 0.9255403600538954]
[1.7179210149514346, 1.2266675316717008, 1.1458094810836168, 1.06664450628627
57, 1.013679662044866, 1.012759315283707, 1.0172652853206696, 1.0419051229633
414, 1.0505210867459562, 1.0505210867459562, 1.0282547175786527, 1.0112995602
427852, 0.9869095410933219, 0.971364834537554]
```



(j)Based on the best test performance, what is your model?

In [32]:
```
print('Conclusion: Based on the figure above increasing of lamda is decreasing the difference between training error and test error. Based on that we can say lamda 2 is the best model from this analysis')
```

```
Conclusion: Based on the figure above increasing of lamda is decreasing the d
ifference between training error and test error. Based on that we can say lam
da 2 is the best model from this analysis
```