1.1 Example Polynomial Curve Fitting

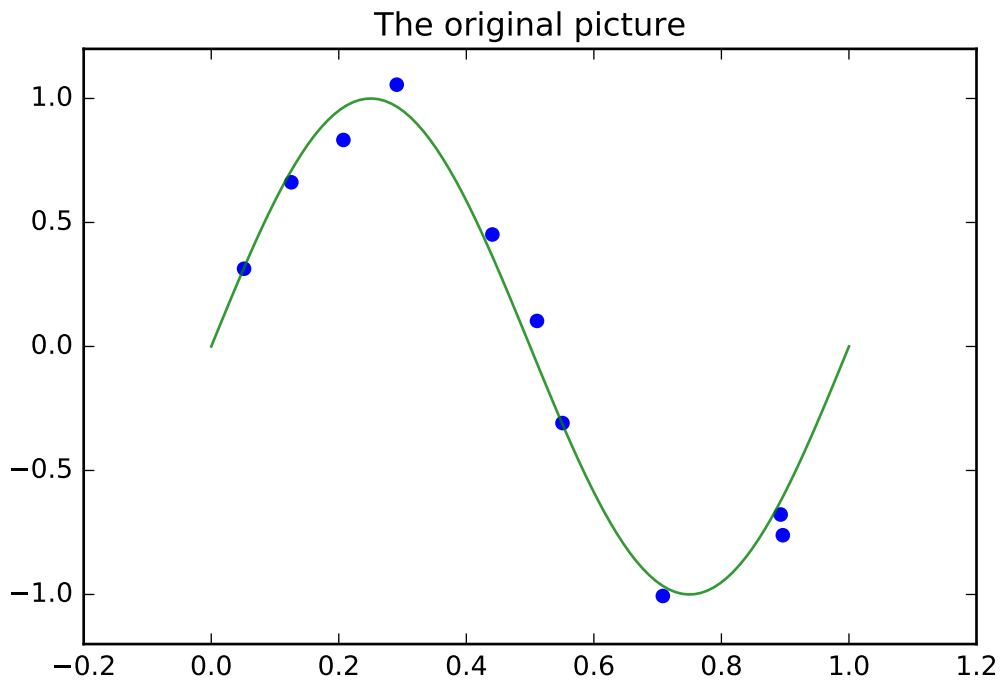
August 17, 2016

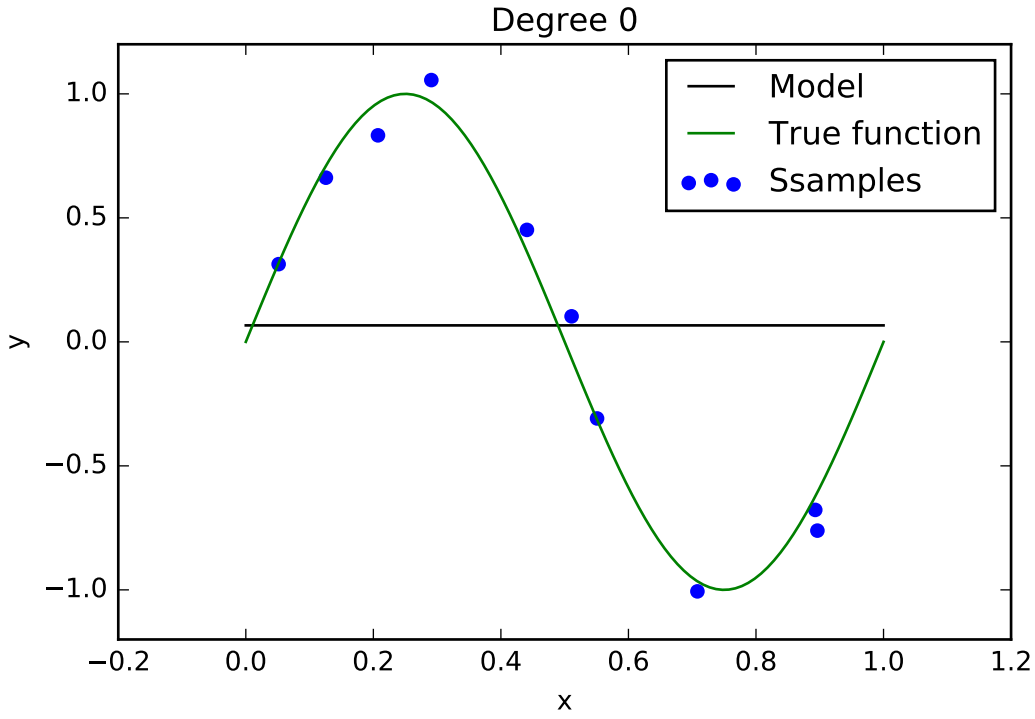
In [1]:
```
#Import some module and set the matplotlib format
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn import cross_validation
from sklearn.learning_curve import validation_curve
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('pdf')
np.random.seed(3)
```

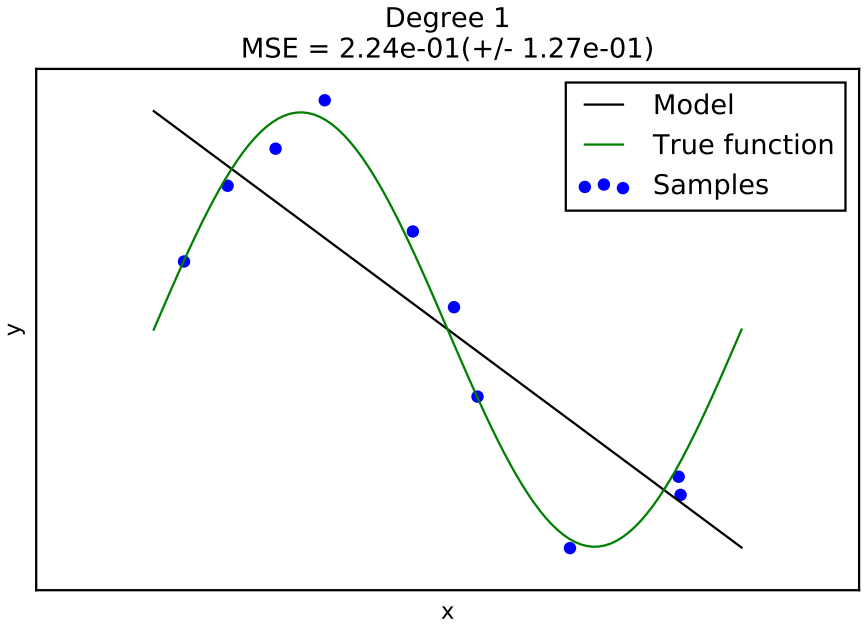
0.1 Figure 1.2,Figure 1.4 Underfitting and Overfitting

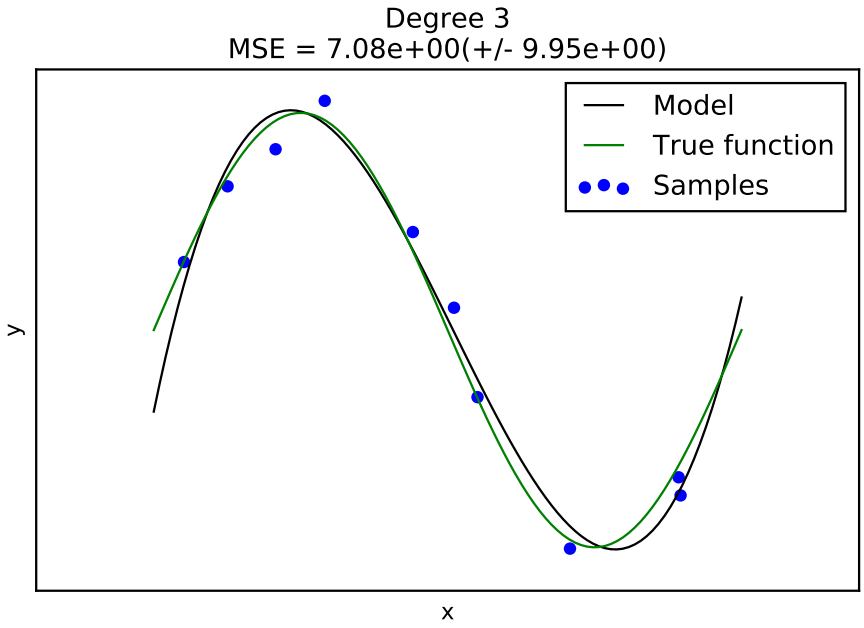
In [2]:
```
n_samples = 10
degrees = [1, 3, 9]

true_fun = lambda X: np.sin(2 * np.pi * X)
X = np.sort(np.random.rand(n_samples))
y = true_fun(X) + np.random.randn(n_samples) * 0.1


#The original picture
X_axis = np.linspace(0, 1, 100)
plt.scatter(X, y, color = 'blue', marker = 'o')
plt.plot(X_axis, true_fun(X_axis), color = 'green', alpha = 0.8)
plt.xlim([-0.2, 1.2])
plt.ylim([-1.2,1.2])
plt.title('The original picture')
plt.show()

#M = 0
plt.xticks , plt.yticks = [], []
plt.scatter(X, y, color = 'blue', marker = 'o', label = 'Ssamples')
p = np.polyfit(X, y, 0)
yfit = np.polyval(p, X_axis)
```

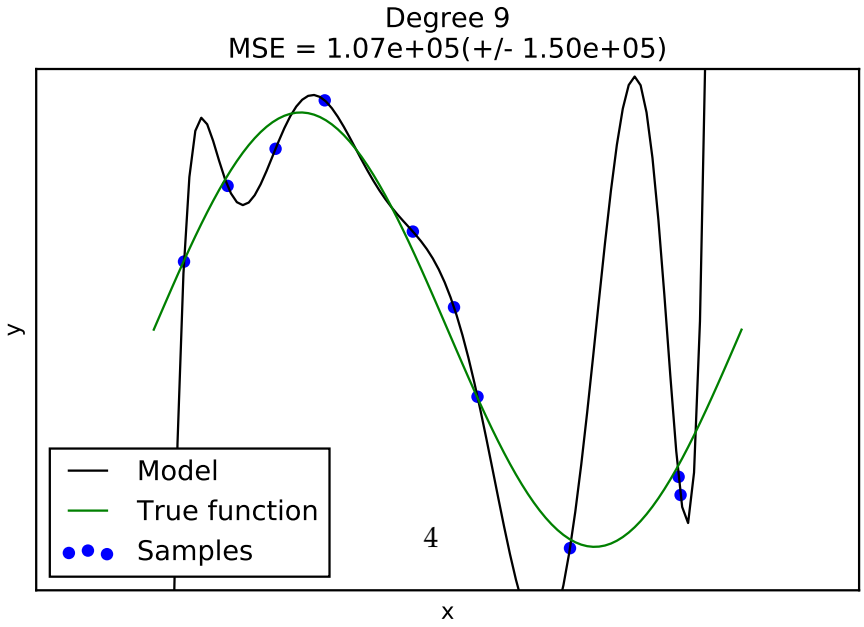
```
plt.plot(X_axis, yfit, color = 'k', label = 'Model')
plt.plot(X_axis, true_fun(X_axis), color = 'green', label = 'True function'
plt.xlim([-0.2, 1.2])
plt.ylim([-1.2, 1.2])
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc = 'best')
plt.title('Degree 0')
plt.show()

plt.figure(figsize=(6.5, 14))
for i in range(len(degrees)):
    ax = plt.subplot(len(degrees), 1, i + 1)
    plt.setp(ax, xticks=(), yticks=())

    polynomial_features = PolynomialFeatures(degree=degrees[i],
                                             include_bias=False)
    linear_regression = LinearRegression()
    pipeline = Pipeline([("polynomial_features", polynomial_features),
                         ("linear_regression", linear_regression)])
    pipeline.fit(X[:, np.newaxis], y)

    # Evaluate the models using crossvalidation
    scores = cross_validation.cross_val_score(pipeline,
        X[:, np.newaxis], y, scoring="mean_squared_error", cv=3)

    X_test = np.linspace(0, 1, 100)
    plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model'
    plt.plot(X_test, true_fun(X_test), label="True function", color = 'gree
    plt.scatter(X, y, label="Samples", color = 'blue')
    plt.xlabel("x")
    plt.ylabel("y")
    plt.xlim((-0.2, 1.2))
    plt.ylim((-1.2, 1.2))
    plt.legend(loc="best")
    plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
        degrees[i], -scores.mean(), scores.std()))
plt.show()
```


The original picture
1.0
0.5
0.0
−0.5
−1.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Degree 0
Model
True function
Ssamples
y
x
1.0
0.5
0.0
−0.5
−1.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Degree 1
MSE = 2.24e-01(+/- 1.27e-01)
Model
True function
Samples
y
x


Degree 3
MSE = 7.08e+00(+/- 9.95e+00)
Model
True function
Samples
y
x


Degree 9
MSE = 1.07e+05(+/- 1.50e+05)
Model
True function
Samples
y
x

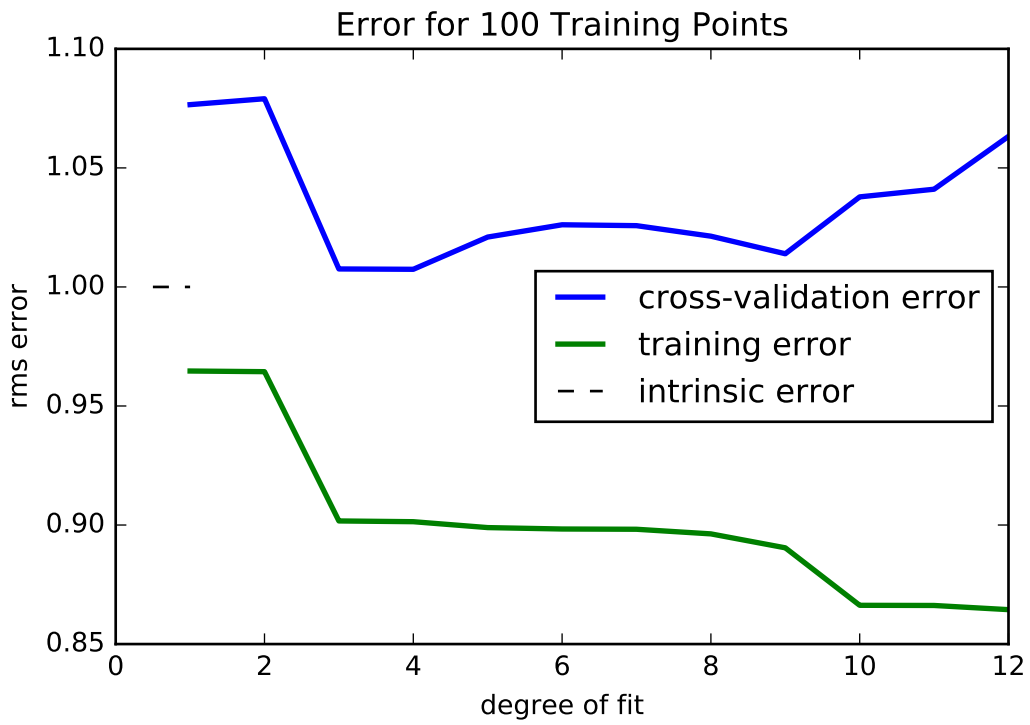
0.2 Figure 1.5 Validation Curve

In [3]:
```
def test_func(x, err = 0.5):
    return np.random.normal(np.sin(2 * np.pi * x), err)
```

In [4]:
```
def compute_error(x, y, p):
    yfit = np.polyval(p, x)
    return np.sqrt((y - yfit).dot((y - yfit).T) / len(y))
```

In [5]:
```
Ntrain = 100
Ncrossval = 100
error = 1.0

np.random.seed(0)
x = np.random.random(Ntrain + Ncrossval)
y = test_func(x, error)

xtrain = x[:Ntrain]
ytrain = y[:Ntrain]

xcrossval = x[Ntrain:]
ycrossval = y[Ntrain:]

degrees = np.arange(1, 13)
train_err = np.zeros(len(degrees))
crossval_err = np.zeros(len(degrees))

for i, d in enumerate(degrees):
    p = np.polyfit(xtrain, ytrain, d)

    train_err[i] = compute_error(xtrain, ytrain, p)
    crossval_err[i] = compute_error(xcrossval, ycrossval, p)

plt.figure()
plt.title('Error for 100 Training Points')
plt.plot(degrees, crossval_err, lw=2, label = 'cross-validation error')
plt.plot(degrees, train_err, lw=2, label = 'training error')
plt.plot([0.5, 1], [error, error], '--k', label='intrinsic error')
plt.legend(loc = 'best')
plt.xlabel('degree of fit')
plt.ylabel('rms error')
```

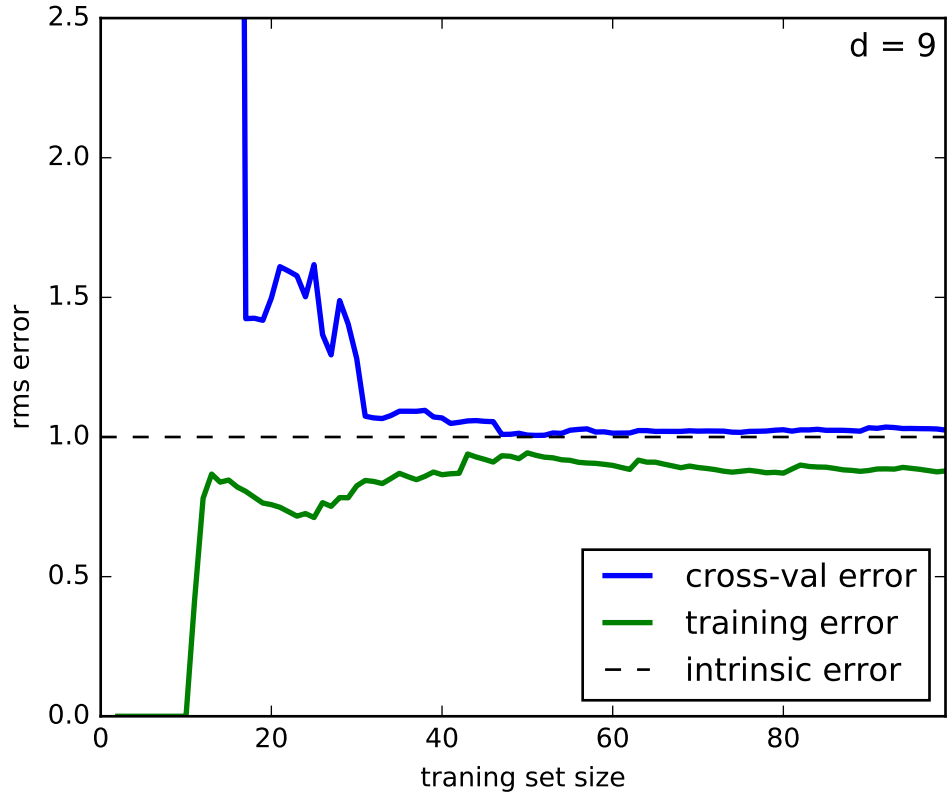
Out[5]:
```
<matplotlib.text.Text at 0x7fefb1150810>
```

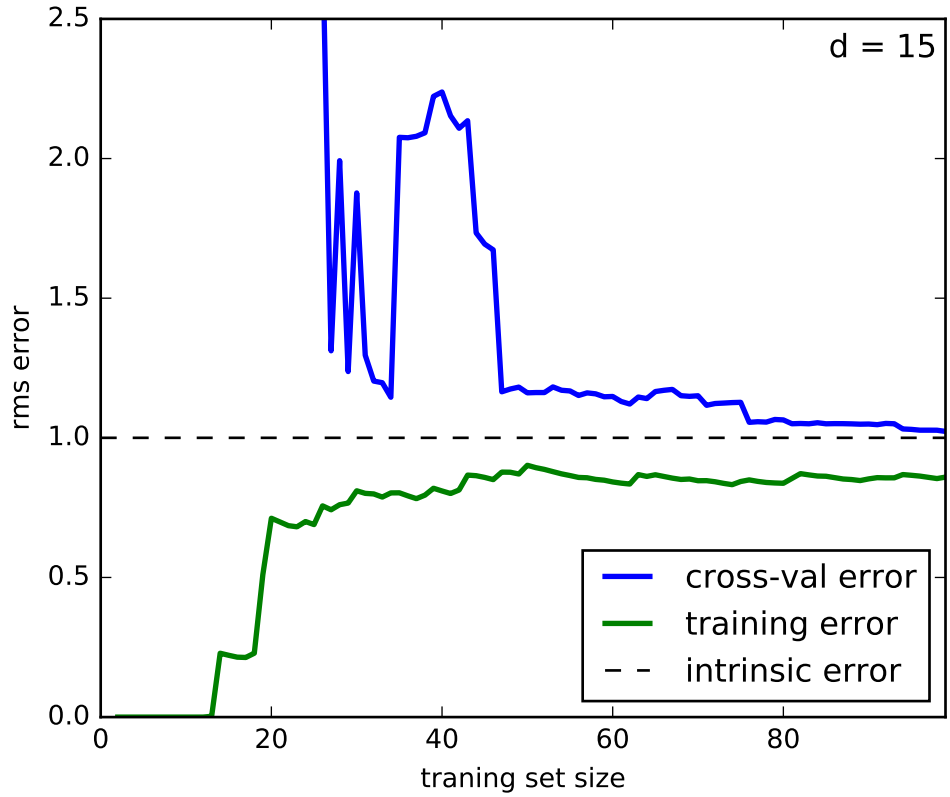


0.3 Increasing the size of data set reduces the over-fitting problem

In [6]:
```
import warnings
warnings.filterwarnings('ignore')#eliminate the bothering warnings

#Specify the number of training set and test set
#as well as thevariability of observed data
Ntrain = 100
Ncrossval = 100
error = 1.0

#Generate some data,sepecify the random seed for reproducing the result
np.random.seed(0)
x = np.random.random(Ntrain + Ncrossval)
y = test_func(x, error)

#Splitting data set
xtrain = x[:Ntrain]
ytrain = y[:Ntrain]

xcrossval = x[Ntrain:]
ycrossval = y[Ntrain:]
```

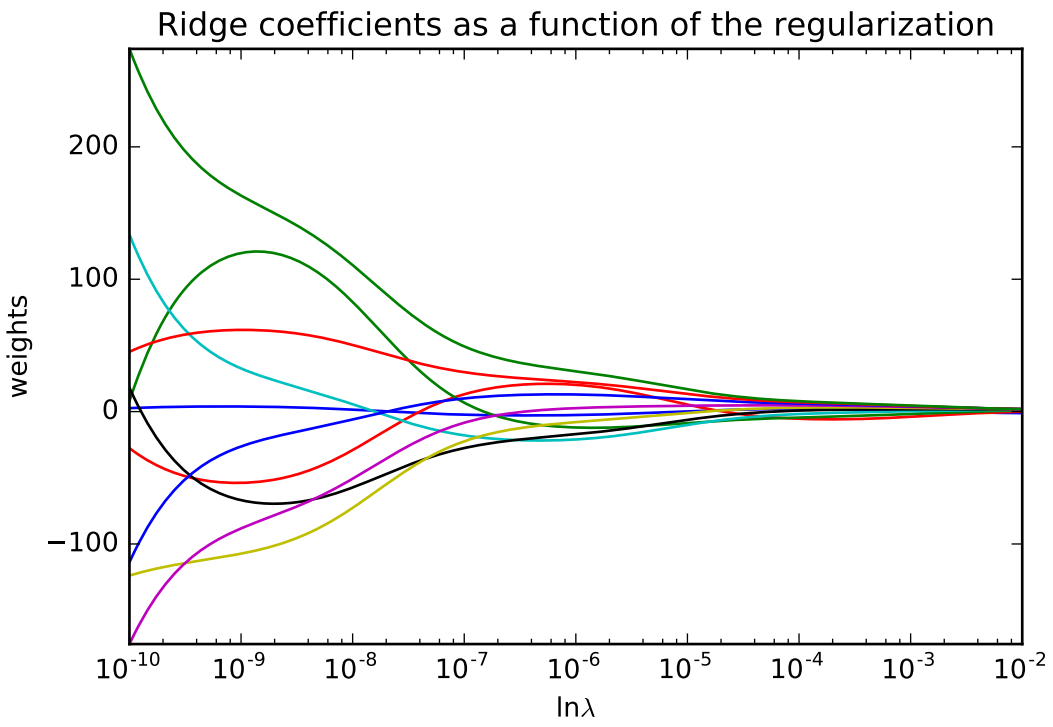
```
sizes = np.linspace(2, Ntrain, 100).astype(int)

train_err = np.zeros(sizes.shape)
crossval_err = np.zeros(sizes.shape)

plt.figure(figsize=(5, 10))

for j,d in enumerate((9, 15)):
    for i, size in enumerate(sizes):
        p = np.polyfit(xtrain[:size], ytrain[:size], d)
        crossval_err[i] = compute_error(xcrossval, ycrossval, p)
        train_err[i] = compute_error(xtrain[:size], ytrain[:size], p)

    ax = plt.subplot(211 + j)
    plt.plot(sizes, crossval_err, lw=2, label='cross-val error')
    plt.plot(sizes, train_err, lw=2, label='training error')
    plt.plot([0, Ntrain], [error, error], '--k', label='intrinsic error')

    plt.xlabel('traning set size')
    plt.ylabel('rms error')
    plt.legend(loc = 'lower right')

    plt.ylim(0.0, 2.5)
    plt.xlim(0, 99)

    plt.text(98, 2.45, 'd = %i' % d, ha='right', va='top', fontsize='large'

plt.subplots_adjust(wspace = 0.02, left=0.07, right=0.95,
                    bottom=0.1, top=0.9)
plt.show()
```


d = 9
rms error
traning set size
cross-val error
training error
intrinsic error
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80


d = 15
rms error
traning set size
cross-val error
training error
intrinsic error
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80

0.4 Add L2 Regularization Term to Reduce The Magnitude of The Coefficients

In [7]:
```
from sklearn import linear_model

# X is the 10x10 Hilbert matrix
X = 1. / (np.arange(1, 11) + np.arange(0, 10)[:, np.newaxis])
y = np.ones(10)

n_alphas = 200
alphas = np.logspace(-10, -2, n_alphas)
clf = linear_model.Ridge(fit_intercept=False)

coefs = []
for a in alphas:
    clf.set_params(alpha=a)
    clf.fit(X, y)
    coefs.append(clf.coef_)

# Display results
ax = plt.gca()
ax.set_color_cycle(['b', 'r', 'g', 'c', 'k', 'y', 'm'])

ax.plot(alphas, coefs)
ax.set_xscale('log')
plt.xlabel('ln$\lambda$')
plt.ylabel('weights')
plt.title('Ridge coefficients as a function of the regularization')
plt.axis('tight')
plt.show()
```



We see that, as the value of λ increases, the typical magitude of the coefficients gets smaller

In [ ]: