In [2]:
```
import numpy as np
import csv
import matplotlib.pyplot as plt
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import Ridge
from scipy import optimize
import pylab as py
from scipy.stats import norm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from scipy.stats import norm
import matplotlib.mlab as mlab
from sklearn.model_selection import StratifiedShuffleSplit
from matplotlib.pyplot import plot
import matplotlib
```

In [3]:
```
# Rebin function taken from http://scipy-cookbook.readthedocs.io/items/R
ebinning.html
def rebin( a, newshape ):
        '''Rebin an array to a new shape.
        '''
        assert len(a.shape) == len(newshape)

        slices = [ slice(0,old, float(old)/new) for old,new in zip(a.sha
pe,newshape) ]
        coordinates = np.mgrid[slices]
        indices = coordinates.astype('i')   #choose the biggest smaller
integer index
        return a[tuple(indices)]

 # Add the relative noise
def add_noise(a,rel_noise,len_data,width_data):
    b = np.ones((len_data,width_data+2))
    for i in range(len(a[:,0])):
        b[i,:] = a[i,:]+ np.random.normal(0, rel_noise,len(a[0,:]))
    return b
```

In [4]:
```
def ProfPSF(amplitudes,positions,len_data,width_data,sigmaPSF, rebin=1):
    profdata=np.ones((len_data, width_data))
    profdataout=np.ones((len_data, int(width_data/rebin)))
    profred=np.ones((rebin,int(width_data/rebin)))
    sigPSF=sigmaPSF#125.0e-6  # [m] original range of the profile data i
s -5...5 mm
#    gaussPSF=[]
    gaussPSF = np.exp(-(positions/sigPSF)**2/2)
   # plt.figure(100)
    #plt.plot(gaussPSF)
#    for p in positions:
#        gaussPSF.append(np.exp(-(p/sigPSF)**2/2))
    # convolution before rebinning:
    for i in range(len_data):
        profdata[i,:]=np.convolve(amplitudes[i,:], gaussPSF, mode="same"
)
       # plt.figure(99)
        #plt.plot(profdata[i,:])
        profred=(profdata[i,:].reshape(int(width_data/rebin),rebin)) # i
n case rebinning is needed
        profdataout[i,:] = profred.mean(axis=1)
        profdataout[i,:] = profdataout[i,:]/np.amax(profdataout[i,:])
       # plt.figure(101)
        #plt.plot(profdataout[i,:])
#            print ("Length rebinned",len(profred))
#            print (profred)
#            profmax=np.amax(profred)
#            print (profmax)
#            profdataout[i,:]=float(np.squeeze(profred))
    return profdataout
```

In [7]:
```
noise = 0.005
PSF = 125.0e-6
pos = np.arange(-5.0e-3,5.0e-3,10.0e-6)
# This comma separated file is matrix of 375*1002
file_name = 'input_train.txt'
sample_num = 375
profile_length = 1000
input_data = np.ones((sample_num,profile_length+2)) # 2 accounts for Np
and sigma_l
rebinning = 10
rebinned_data = np.ones((sample_num,int(profile_length/rebinning)+2))
noisy_rebinned_data  = np.ones((sample_num,int(profile_length/rebinning)
+2))
```

In [8]:
```
with open(file_name, 'r') as f:
    reader = csv.reader(f,delimiter=',')
    rownum = 0
    for row in reader:
#        print (row)
        colnum = 0
        for col in row:
            input_data[rownum,colnum] = float(col)
            colnum = colnum +1
        rownum = rownum+1
```

In [9]:
```
# Only profiele data for rebinning
profile_data = input_data[:,0:1000]

#plt.figure(5)
#plt.plot(profile_data[10,:])
# Rebin the data
# Do the PSF here
rebinned_data = ProfPSF(profile_data,pos,sample_num,profile_length,PSF,r
ebinning)
#rebinned_data = rebin(profile_data,[375,int(profile_length/rebinning)])

# Do the PSF here

# Add Np and sigma_l back
rebinned_data = np.concatenate((rebinned_data,input_data[:,1000:1002]),1
)

# Add noise
noisy_rebinned_data = add_noise(rebinned_data,noise,sample_num,int(profi
le_length/rebinning))
#plt.plot(noisy_rebinned_data[10,0:1000])
#plt.plot(rebinned_data[10,:])
#plt.plot(noisy_rebinned_data[10,:])

#plt.show()


#print('rebinned data',rebinned_data)
#print('noisy_rebinned_data',noisy_rebinned_data)
print('noisy_rebinned_data ',len(noisy_rebinned_data[0]) )
```

```
noisy_rebinned_data  102
```

In [11]:
```
matrix2=[]

myfile2= open('output_train.txt','r')
for line in myfile2.readlines():
    for i in line.split(","):
        matrix2.append(i)


Ytrain=np.squeeze(np.asarray(matrix2))
y0=np.array(Ytrain).astype(np.float)
print('y0',len(y0))
```

```
y0 375
```

In [12]:
```
# This comma separated file is matrix of 375*1002
file_name = 'input_val.txt'
sample_num = 128
input_data50 = np.ones((sample_num,profile_length+2)) # 2 accounts for N
p and sigma_l
rebinned_data50 = np.ones((sample_num,int(profile_length/rebinning)+2))
noisy_rebinned_data50  = np.ones((sample_num,int(profile_length/rebinnin
g)+2))
```

In [13]:
```
with open(file_name, 'r') as f:
    reader = csv.reader(f,delimiter=',')
    rownum = 0
    for row in reader:
#         print (row)
        colnum = 0
        for col in row:
            input_data50[rownum,colnum] = float(col)
            colnum = colnum +1
        rownum = rownum+1

# Only profiele data for rebinning
profile_data50 = input_data50[:,0:1000]

# Rebin the data
rebinned_data50 = ProfPSF(profile_data50,pos,sample_num,profile_length,P
SF,rebinning)
#rebinned_data50 = rebin(profile_data50,[sample_num,int(profile_length/r
ebinning)])
#function rebin is defined already
# Add Np and sigma_l back
rebinned_data50 = np.concatenate((rebinned_data50,input_data50[:,1000:10
02]),1)
#noise = 0.0
# Add noise
noisy_rebinned_data50 = add_noise(rebinned_data50,noise,sample_num,int(p
rofile_length/rebinning))

#plt.plot(rebinned_data50[10,:])
#plt.plot(noisy_rebinned_data50[10,:])

#plt.show()


#print('rebinned data50',rebinned_data50)
#print('noisy_rebinned_data50',noisy_rebinned_data50)
```

In [14]:
```
matrix50=[]

myfile50= open('output_val.txt','r')
for line in myfile50.readlines():
    for i in line.split(","):
        matrix50.append(i)


Ytrain50=np.squeeze(np.asarray(matrix50))
y50=np.array(Ytrain50).astype(np.float)
print('y50',len(y50))
#x=noisy_rebinned_data50
#y=y50

#25%validation data
```

```
y50 128
```

In [15]:
```
# This comma separated file is matrix of 375*1002
file_name = 'input_val.txt'
sample_num = 128
input_data25 = np.ones((sample_num,profile_length+2)) # 2 accounts for N
p and sigma_l
rebinned_data25 = np.ones((sample_num,int(profile_length/rebinning)+2))
noisy_rebinned_data25  = np.ones((sample_num,int(profile_length/rebinnin
g)+2))


with open(file_name, 'r') as f:
    reader = csv.reader(f,delimiter=',')
    rownum = 0
    for row in reader:
#        print (row)
        colnum = 0
        for col in row:
            input_data25[rownum,colnum] = float(col)
            colnum = colnum +1
        rownum = rownum+1

# Only profiele data for rebinning
profile_data25 = input_data25[:,0:1000]

# Rebin the data
rebinned_data25 = ProfPSF(profile_data25,pos,sample_num,profile_length,P
SF,rebinning)
#rebinned_data25 = rebin(profile_data25,[128,int(profile_length/rebinnin
g)])

# Add Np and sigma_l back
rebinned_data25 = np.concatenate((rebinned_data25,input_data25[:,1000:10
02]),1)
#noise = 0.01
# Add noise
noisy_rebinned_data25 = add_noise(rebinned_data25,noise,sample_num,int(p
rofile_length/rebinning))

#plt.plot(rebinned_data50[10,:])
#plt.plot(noisy_rebinned_data50[10,:])

#plt.show()


#print('rebinned data50',rebinned_data50)
#print('noisy_rebinned_data50',noisy_rebinned_data50)
```

In [17]:
```
matrix25=[]

myfile25= open('output_val.txt','r')
for line in myfile25.readlines():
    for i in line.split(","):
        matrix25.append(i)


Ytrain25=np.squeeze(np.asarray(matrix25))
y25=np.array(Ytrain25).astype(np.float)
print('y25',len(y25))
#x=noisy_rebinned_data25
#y=y25



#1% validation data
```

```
y25 128
```

In [19]:

```
# This comma separated file is matrix of 375*1002
file_name = 'input_val.txt'
sample_num = 128
input_data01 = np.ones((sample_num,profile_length+2)) # 2 accounts for N
p and sigma_l
rebinned_data01 = np.ones((sample_num,int(profile_length/rebinning)+2))
noisy_rebinned_data01  = np.ones((sample_num,int(profile_length/rebinnin
g)+2))


with open(file_name, 'r') as f:
    reader = csv.reader(f,delimiter=',')
    rownum = 0
    for row in reader:
#        print (row)
        colnum = 0
        for col in row:
            input_data01[rownum,colnum] = float(col)
            colnum = colnum +1
        rownum = rownum+1

# Only profiele data for rebinning
profile_data01 = input_data01[:,0:1000]

# Rebin the data
rebinned_data01 = ProfPSF(profile_data01,pos,sample_num,profile_length,P
SF,rebinning)
#rebinned_data01 = rebin(profile_data01,[sample_num,int(profile_length/r
ebinning)])

# Add Np and sigma_l back
rebinned_data01 = np.concatenate((rebinned_data01,input_data01[:,1000:10
02]),1)
#noise = 0.01
# Add noise
noisy_rebinned_data01 = add_noise(rebinned_data01,noise,sample_num,int(p
rofile_length/rebinning))

#plt.plot(rebinned_data50[10,:])
#plt.plot(noisy_rebinned_data50[10,:])

#plt.show()


#print('rebinned data50',rebinned_data50)
#print('noisy_rebinned_data50',noisy_rebinned_data50)
```

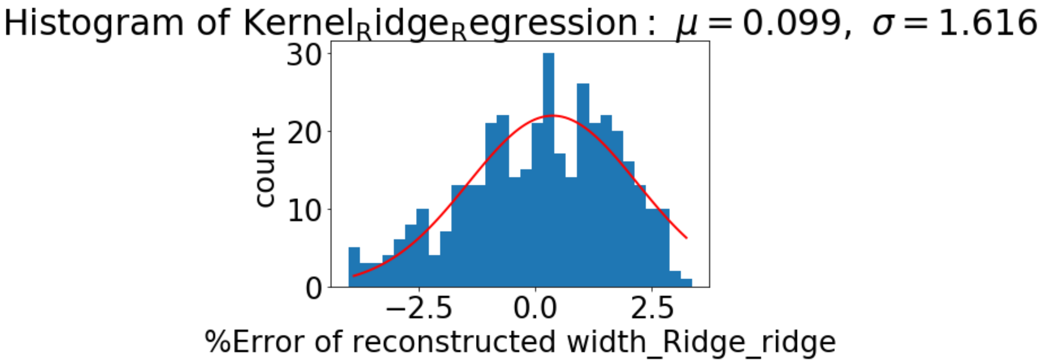
In [21]:

```
matrix01=[]

myfile01= open('output_val.txt','r')
for line in myfile01.readlines():
    for i in line.split(","):
        matrix01.append(i)


Ytrain01=np.squeeze(np.asarray(matrix01))
y01=np.array(Ytrain01).astype(np.float)
print('y01',len(y01))
#x=noisy_rebinned_data01
#y=y01
```

```
y01 128
```

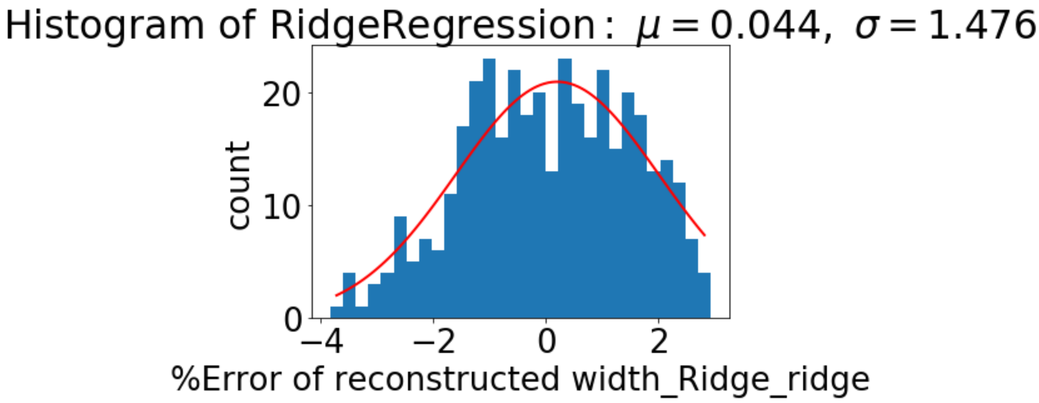
In [40]:
```
#kernel ridge regression:

clf = KernelRidge(alpha=1.0, kernel='linear')
print(clf.fit)
print('w',rebinned_data)
clf.fit(noisy_rebinned_data, y0)

print('KR_regression coef length',len(clf.dual_coef_)) #coefficients of
prediction function
print(clf.fit)

#50% validation data
y_pred50=clf.predict(noisy_rebinned_data50) #prediction function Ridge R
egression
print('y50',len(y50))
Error_K_ridge_regression50=((y50-y_pred50)*100)/(y50)


#25% validation data
y_pred25=clf.predict(noisy_rebinned_data25) #prediction function Ridge R
egression
print('y25',len(y25))
Error_K_ridge_regression25=((y25-y_pred25)*100)/(y25)


#01% validation data
y_pred01=clf.predict(noisy_rebinned_data01) #prediction function Ridge R
egression
print('y01',len(y01))
Error_K_ridge_regression01=((y01-y_pred01)*100)/(y01)

matplotlib.rcParams.update({'font.size': 26}) #coefficients of predictio
n function

#plt.figure(10)
#plt.plot(clf.dual_coef_)
#plt.plot(noisy_rebinned_data[10,:]/5)
#plt.ylabel('weight')
#plt.xlabel('coefficients and original profile')

total_error=[]
total_error=np.concatenate((Error_K_ridge_regression50,Error_K_ridge_reg
ression25,Error_K_ridge_regression01), axis=0)

mu = np.mean(total_error)
sigma = np.std(total_error)



x1 = total_error

plt.figure(27)

data = py.hist(x1, bins = 30)
#Equation for Gaussian
def f(x, a, b, c):
    return a * py.exp(-(x - b*b)**2.0 / (2 * c**2))
# Generate data from bins as a set of points
x = [0.5 * (data[1][i] + data[1][i+1]) for i in range(len(data[1])-1)]
y = data[0]
popt, pcov = optimize.curve_fit(f, x, y)
x_fit = py.linspace(x[0], x[-1], 100)
print('x_fit',x_fit)


y_fit = f(x_fit, *popt) #generates the fitting-curve
print('y_fit',y_fit)
print('constant:a, mean:b, sigma:c',*popt)
```

```
<bound method KernelRidge.fit of KernelRidge(alpha=1.0, coef0=1, degree=3
, gamma=None, kernel='linear',
      kernel_params=None)>
w [[  2.57263727e-137   6.04419396e-129   7.49405765e-121 ...,
    5.78008700e-146   8.75000000e-001   9.11764706e-001]
 [  6.95857486e-156   6.25892641e-147   2.97057423e-138 ...,
    3.41045952e-163   8.75000000e-001   1.00000000e+000]
 [  2.66198872e-162   3.75270537e-153   2.79210599e-144 ...,
    8.05448032e-170   8.75000000e-001   7.35294118e-001]
 ...,
 [  4.23369661e-145   1.76818806e-136   3.89698597e-128 ...,
    6.02054751e-134   7.50000000e-001   7.35294118e-001]
 [  9.27540622e-141   2.81406311e-132   4.50548081e-124 ...,
    3.85714932e-128   7.50000000e-001   8.23529412e-001]
 [  1.41416988e-148   7.68486441e-140   2.20465991e-131 ...,
    4.12235581e-133   7.50000000e-001   8.23529412e-001]]
KR_regression coef length 375
<bound method KernelRidge.fit of KernelRidge(alpha=1.0, coef0=1, degree=3
, gamma=None, kernel='linear',
      kernel_params=None)>
y50 128
y25 128
y01 128
x_fit [ -3.89237182e+00  -3.82032759e+00  -3.74828336e+00  -3.67623913e+0
0
  -3.60419490e+00  -3.53215067e+00  -3.46010644e+00  -3.38806221e+00
  -3.31601798e+00  -3.24397374e+00  -3.17192951e+00  -3.09988528e+00
  -3.02784105e+00  -2.95579682e+00  -2.88375259e+00  -2.81170836e+00
  -2.73966413e+00  -2.66761990e+00  -2.59557567e+00  -2.52353144e+00
  -2.45148721e+00  -2.37944298e+00  -2.30739875e+00  -2.23535452e+00
  -2.16331029e+00  -2.09126606e+00  -2.01922183e+00  -1.94717760e+00
  -1.87513337e+00  -1.80308914e+00  -1.73104491e+00  -1.65900068e+00
  -1.58695645e+00  -1.51491221e+00  -1.44286798e+00  -1.37082375e+00
  -1.29877952e+00  -1.22673529e+00  -1.15469106e+00  -1.08264683e+00
  -1.01060260e+00  -9.38558371e-01  -8.66514141e-01  -7.94469910e-01
  -7.22425680e-01  -6.50381450e-01  -5.78337219e-01  -5.06292989e-01
  -4.34248758e-01  -3.62204528e-01  -2.90160297e-01  -2.18116067e-01
  -1.46071837e-01  -7.40276062e-02  -1.98337577e-03   7.00608547e-02
   1.42105085e-01   2.14149315e-01   2.86193546e-01   3.58237776e-01
   4.30282007e-01   5.02326237e-01   5.74370468e-01   6.46414698e-01
   7.18458928e-01   7.90503159e-01   8.62547389e-01   9.34591620e-01
   1.00663585e+00   1.07868008e+00   1.15072431e+00   1.22276854e+00
   1.29481277e+00   1.36685700e+00   1.43890123e+00   1.51094546e+00
   1.58298969e+00   1.65503392e+00   1.72707815e+00   1.79912238e+00
   1.87116662e+00   1.94321085e+00   2.01525508e+00   2.08729931e+00
   2.15934354e+00   2.23138777e+00   2.30343200e+00   2.37547623e+00
   2.44752046e+00   2.51956469e+00   2.59160892e+00   2.66365315e+00
   2.73569738e+00   2.80774161e+00   2.87978584e+00   2.95183007e+00
   3.02387430e+00   3.09591853e+00   3.16796276e+00   3.24000699e+00]
y_fit [  1.35915875   1.49177477   1.63473648   1.78856068   1.95375917
   2.13083492   2.32027792   2.52256077   2.73813396   2.96742105
   3.2108135    3.46866545   3.74128829   4.0289452    4.33184565
   4.65013983   4.98391328   5.33318155   5.69788508   6.07788432
   6.47295518   6.88278488   7.30696822   7.74500437   8.19629426
   8.66013861   9.13573662   9.62218548  10.11848062  10.62351683
  11.13609027  11.65490134  12.17855849  12.705583    13.23441457
  13.76341796  14.2908904   14.81506989  15.33414434  15.84626139
  16.34953895  16.84207633  17.32196588  17.78730501  18.23620863
  18.66682165  19.07733168  19.46598162  19.83108216  20.17102394
  20.48428936  20.76946384  21.02524646  21.25045979  21.444059
  21.60513985  21.73294579  21.82687381  21.88647925  21.91147927
  21.90175507  21.85735288  21.77848354  21.66552086  21.51899869
  21.33960665  21.12818481  20.88571707  20.61332355  20.312252
  19.98386826  19.62964594  19.25115545  18.8500524   18.42806557
  17.98698457  17.52864727  17.05492711  16.56772051  16.0689344   15.560
474
  15.044231    14.52207215  13.99582854  13.46728537  12.93817254
  12.410156    11.88482986  11.36370951  10.84822549  10.3397184
```


Histogram of Kernel$_{\mathrm{R}}$idge$_{\mathrm{R}}$egression : $\mu = 0.099,\ \sigma = 1.616$
30
20
10
0
count
−2.5
0.0
2.5
%Error of reconstructed width_Ridge_ridge

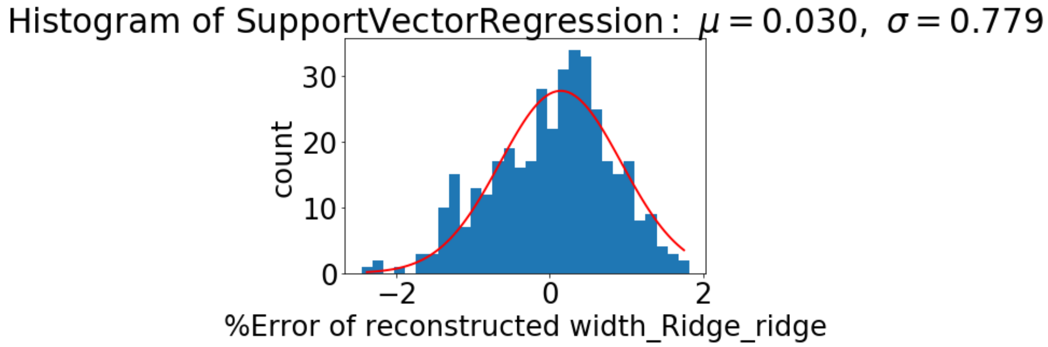
In [42]:
```
#ridge regression:

clf = Ridge(alpha=1.0)
print(clf.fit)
print('w',rebinned_data)
clf.fit(noisy_rebinned_data, y0)

print('Ridge_regression coef length',len(clf.coef_)) #coefficients of pr
ediction function
print(clf.fit)

#50% validation data
y_pred50=clf.predict(noisy_rebinned_data50) #prediction function Ridge R
egression
print('y50',len(y50))
Error_K_ridge_regression50=((y50-y_pred50)*100)/(y50)


#25% validation data
y_pred25=clf.predict(noisy_rebinned_data25) #prediction function Ridge R
egression
print('y25',len(y25))
Error_K_ridge_regression25=((y25-y_pred25)*100)/(y25)


#01% validation data
y_pred01=clf.predict(noisy_rebinned_data01) #prediction function Ridge R
egression
print('y01',len(y01))
Error_K_ridge_regression01=((y01-y_pred01)*100)/(y01)

matplotlib.rcParams.update({'font.size': 26}) #coefficients of predictio
n function

#plt.figure(10)
#plt.plot(clf.dual_coef_)
#plt.plot(noisy_rebinned_data[10,:]/5)
#plt.ylabel('weight')
#plt.xlabel('coefficients and original profile')

total_error=[]
total_error=np.concatenate((Error_K_ridge_regression50,Error_K_ridge_reg
ression25,Error_K_ridge_regression01), axis=0)

mu = np.mean(total_error)
sigma = np.std(total_error)



x1 = total_error

plt.figure(27)

data = py.hist(x1, bins = 30)
#Equation for Gaussian
def f(x, a, b, c):
    return a * py.exp(-(x - b*b)**2.0 / (2 * c**2))
# Generate data from bins as a set of points
x = [0.5 * (data[1][i] + data[1][i+1]) for i in range(len(data[1])-1)]
y = data[0]
popt, pcov = optimize.curve_fit(f, x, y)
x_fit = py.linspace(x[0], x[-1], 100)
print('x_fit',x_fit)


y_fit = f(x_fit, *popt) #generates the fitting-curve
print('y_fit',y_fit)
print('constant:a, mean:b, sigma:c',*popt)
```

```
<bound method Ridge.fit of Ridge(alpha=1.0, copy_X=True, fit_intercept=Tr
ue, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)>
w [[  2.57263727e-137   6.04419396e-129   7.49405765e-121 ...,
    5.78008700e-146   8.75000000e-001   9.11764706e-001]
 [  6.95857486e-156   6.25892641e-147   2.97057423e-138 ...,
    3.41045952e-163   8.75000000e-001   1.00000000e+000]
 [  2.66198872e-162   3.75270537e-153   2.79210599e-144 ...,
    8.05448032e-170   8.75000000e-001   7.35294118e-001]
 ...,
 [  4.23369661e-145   1.76818806e-136   3.89698597e-128 ...,
    6.02054751e-134   7.50000000e-001   7.35294118e-001]
 [  9.27540622e-141   2.81406311e-132   4.50548081e-124 ...,
    3.85714932e-128   7.50000000e-001   8.23529412e-001]
 [  1.41416988e-148   7.68486441e-140   2.20465991e-131 ...,
    4.12235581e-133   7.50000000e-001   8.23529412e-001]]
Ridge_regression coef length 102
<bound method Ridge.fit of Ridge(alpha=1.0, copy_X=True, fit_intercept=Tr
ue, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)>
y50 128
y25 128
y01 128
x_fit [-3.72356675 -3.65739444 -3.59122213 -3.52504982 -3.45887751 -3.392
7052
 -3.32653289 -3.26036058 -3.19418826 -3.12801595 -3.06184364 -2.99567133
 -2.92949902 -2.86332671 -2.7971544  -2.73098208 -2.66480977 -2.59863746
 -2.53246515 -2.46629284 -2.40012053 -2.33394822 -2.2677759  -2.20160359
 -2.13543128 -2.06925897 -2.00308666 -1.93691435 -1.87074204 -1.80456973
 -1.73839741 -1.6722251  -1.60605279 -1.53988048 -1.47370817 -1.40753586
 -1.34136355 -1.27519123 -1.20901892 -1.14284661 -1.0766743  -1.01050199
 -0.94432968 -0.87815737 -0.81198505 -0.74581274 -0.67964043 -0.61346812
 -0.54729581 -0.4811235  -0.41495119 -0.34877888 -0.28260656 -0.21643425
 -0.15026194 -0.08408963 -0.01791732  0.04825499  0.1144273   0.18059962
  0.24677193  0.31294424  0.37911655  0.44528886  0.51146117  0.57763348
  0.6438058   0.70997811  0.77615042  0.84232273  0.90849504  0.97466735
  1.04083966  1.10701197  1.17318429  1.2393566   1.30552891  1.37170122
  1.43787353  1.50404584  1.57021815  1.63639047  1.70256278  1.76873509
  1.8349074   1.90107971  1.96725202  2.03342433  2.09959664  2.16576896
  2.23194127  2.29811358  2.36428589  2.4304582   2.49663051  2.56280282
  2.62897514  2.69514745  2.76131976  2.82749207]
y_fit [  1.97028116   2.13197158   2.30384396   2.48624061   2.67948727
   2.88388995   3.09973178   3.3272696    3.56673065   3.81830909
   4.08216255   4.35840869   4.64712179   4.94832941   5.26200912
   5.5880854    5.92642672   6.27684274   6.63908185   7.01282889
   7.39770327   7.79325735   8.19897521   8.61427187   9.03849288
   9.4709144    9.91074377  10.35712052  10.80911798  11.26574533
  11.72595024  12.18862203  12.65259533  13.11665427  13.57953723
  14.03994201  14.49653148  14.94793971  15.3927784   15.82964379
  16.25712377  16.67380537  17.07828231  17.46916289  17.84507779
  18.20468808  18.54669302  18.86983796  19.17292189  19.45480493
  19.7144154   19.95075665  20.16291339  20.35005761  20.51145397
  20.6464646   20.75455328  20.83528904  20.88834898  20.91352041
  20.91070234  20.87990606  20.82125516  20.73498461  20.62143928
  20.48107158  20.31443848  20.12219783  19.90510403  19.66400312
  19.39982723  19.11358866  18.80637336  18.4793341   18.13368329
  17.77068549  17.39164973  16.99792172  16.5908759   16.17190755
  15.74242491  15.30384139  14.85756802  14.40500599  13.94753965
  13.48652965  13.02330656  12.55916485  12.09535734  11.63309008
  11.17351778  10.71773963  10.26679584   9.8216645    9.38325911
   8.95242659   8.52994582   8.11652672   7.71280973   7.3193659 ]
constant:a, mean:b, sigma:c 20.9157538149 -0.455001681828 1.80831114614
```


Histogram of RidgeRegression: $\mu = 0.044,\ \sigma = 1.476$
count
20
10
0
−4
−2
0
2
%Error of reconstructed width_Ridge_ridge

In [48]:
```
#Using SVM for regression (SVR) :


clf = SVR(C=10, cache_size=200, coef0=0.0, degree=3, epsilon=0.01, gamma
=0.01, \
        kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=Fa
lse)
#print('SVR_regression_coefficients',clf.dual_coef_)

print(clf.fit)
print('w',rebinned_data)
clf.fit(noisy_rebinned_data, y0)

print('KR_regression coef length',len(clf.dual_coef_)) #coefficients of
prediction function
print(clf.fit)

#50% validation data
y_pred50=clf.predict(noisy_rebinned_data50) #prediction function Ridge R
egression
print('y50',len(y50))
Error_K_ridge_regression50=((y50-y_pred50)*100)/(y50)


#25% validation data
y_pred25=clf.predict(noisy_rebinned_data25) #prediction function Ridge R
egression
print('y25',len(y25))
Error_K_ridge_regression25=((y25-y_pred25)*100)/(y25)


#01% validation data
y_pred01=clf.predict(noisy_rebinned_data01) #prediction function Ridge R
egression
print('y01',len(y01))
Error_K_ridge_regression01=((y01-y_pred01)*100)/(y01)

matplotlib.rcParams.update({'font.size': 26}) #coefficients of predictio
n function

#plt.figure(10)
#plt.plot(clf.dual_coef_)
#plt.plot(noisy_rebinned_data[10,:]/5)
#plt.ylabel('weight')
#plt.xlabel('coefficients and original profile')

total_error=[]
total_error=np.concatenate((Error_K_ridge_regression50,Error_K_ridge_reg
ression25,Error_K_ridge_regression01), axis=0)

mu = np.mean(total_error)
sigma = np.std(total_error)



x1 = total_error

plt.figure(27)

data = py.hist(x1, bins = 30)
#Equation for Gaussian
def f(x, a, b, c):
    return a * py.exp(-(x - b*b)**2.0 / (2 * c**2))
# Generate data from bins as a set of points
x = [0.5 * (data[1][i] + data[1][i+1]) for i in range(len(data[1])-1)]
y = data[0]
popt, pcov = optimize.curve_fit(f, x, y)
x_fit = py.linspace(x[0], x[-1], 100)
```

```
<bound method BaseLibSVM.fit of SVR(C=10, cache_size=200, coef0=0.0, degr
ee=3, epsilon=0.01, gamma=0.01,
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)>
w [[  2.57263727e-137   6.04419396e-129   7.49405765e-121 ...,
    5.78008700e-146   8.75000000e-001   9.11764706e-001]
 [  6.95857486e-156   6.25892641e-147   2.97057423e-138 ...,
    3.41045952e-163   8.75000000e-001   1.00000000e+000]
 [  2.66198872e-162   3.75270537e-153   2.79210599e-144 ...,
    8.05448032e-170   8.75000000e-001   7.35294118e-001]
 ...,
 [  4.23369661e-145   1.76818806e-136   3.89698597e-128 ...,
    6.02054751e-134   7.50000000e-001   7.35294118e-001]
 [  9.27540622e-141   2.81406311e-132   4.50548081e-124 ...,
    3.85714932e-128   7.50000000e-001   8.23529412e-001]
 [  1.41416988e-148   7.68486441e-140   2.20465991e-131 ...,
    4.12235581e-133   7.50000000e-001   8.23529412e-001]]
KR_regression coef length 1
<bound method BaseLibSVM.fit of SVR(C=10, cache_size=200, coef0=0.0, degr
ee=3, epsilon=0.01, gamma=0.01,
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)>
y50 128
y25 128
y01 128
x_fit [-2.38881189 -2.34698372 -2.30515555 -2.26332737 -2.2214992  -2.179
67103
 -2.13784286 -2.09601469 -2.05418652 -2.01235835 -1.97053018 -1.92870201
 -1.88687384 -1.84504567 -1.8032175  -1.76138933 -1.71956115 -1.67773298
 -1.63590481 -1.59407664 -1.55224847 -1.5104203  -1.46859213 -1.42676396
 -1.38493579 -1.34310762 -1.30127945 -1.25945128 -1.21762311 -1.17579493
 -1.13396676 -1.09213859 -1.05031042 -1.00848225 -0.96665408 -0.92482591
 -0.88299774 -0.84116957 -0.7993414  -0.75751323 -0.71568506 -0.67385689
 -0.63202871 -0.59020054 -0.54837237 -0.5065442  -0.46471603 -0.42288786
 -0.38105969 -0.33923152 -0.29740335 -0.25557518 -0.21374701 -0.17191884
 -0.13009067 -0.08826249 -0.04643432 -0.00460615  0.03722202  0.07905019
  0.12087836  0.16270653  0.2045347   0.24636287  0.28819104  0.33001921
  0.37184738  0.41367555  0.45550372  0.4973319   0.53916007  0.58098824
  0.62281641  0.66464458  0.70647275  0.74830092  0.79012909  0.83195726
  0.87378543  0.9156136   0.95744177  0.99926994  1.04109812  1.08292629
  1.12475446  1.16658263  1.2084108   1.25023897  1.29206714  1.33389531
  1.37572348  1.41755165  1.45937982  1.50120799  1.54303616  1.58486434
  1.62669251  1.66852068  1.71034885  1.75217702]
y_fit [  0.16581229   0.19610332   0.23128062   0.2720067    0.31901127
   0.37309421   0.43512802   0.50605959   0.58691112   0.67878005
   0.78283795   0.90032795   1.03256085   1.18090957   1.34680181
   1.5317109    1.73714462   1.964632     2.21570794   2.49189579
   2.79468782   3.12552366   3.4857669    3.87668002   4.29939782
   4.75489982   5.24398178   5.7672269    6.32497722   6.91730549
   7.54398835   8.20448121   8.89789555   9.62297914  10.37809993
  11.16123404  11.96995848  12.80144917  13.65248446  14.51945478
  15.39837844  16.28492384  17.17443812  18.06198204  18.94237095
  19.81022148  20.66000336  21.48609577  22.28284756  23.04464032
  23.76595343  24.44143002  25.06594285  25.63465877  26.14310092
  26.58720723  26.96338449  27.26855667  27.50020687  27.65641179
  27.73586839  27.73791197  27.66252541  27.51033944  27.28262382
  26.98126972  26.60876357  26.1681529   25.66300482  25.09735796
  24.47566877  23.80275312  23.08372436  22.32392888  21.52888032
  20.70419357  19.85551966  18.98848253  18.10861867  17.22132054
  16.33178446  15.4449637   14.56552715  13.6978241   12.84585529
  12.01325037  11.20325173  10.41870456   9.66205298   8.93534177
   8.24022339   7.57796971   6.94948797   6.35534034   5.7957665
   5.27070866   4.77983829   4.32258418   3.89816106   3.50559839]
constant:a, mean:b, sigma:c 27.7466100434 -0.378014516514 0.791157420439
```


Histogram of SupportVectorRegression: $\mu = 0.030,\ \sigma = 0.779$
count
30
20
10
0
−2
0
2
%Error of reconstructed width_Ridge_ridge