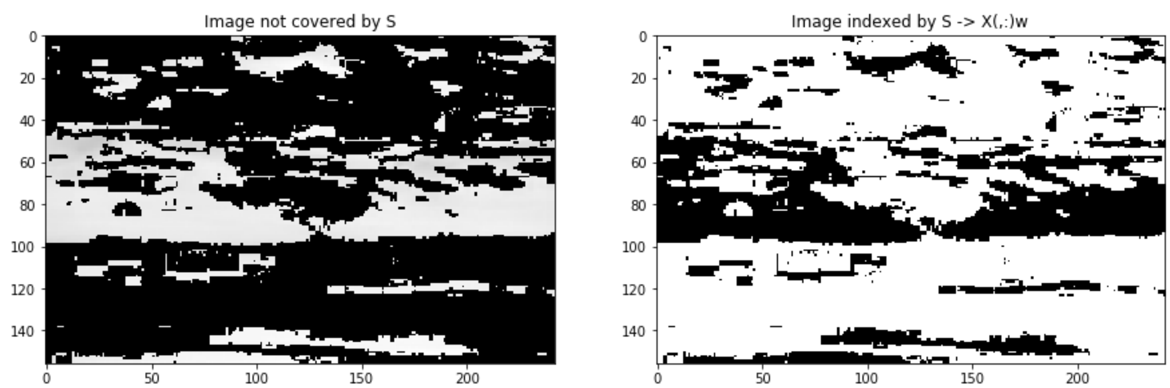
In [69]:
```
y_1 = X @ w
y_2 = y_1.copy()

for i in S:
    y_1[i] = 0
    y_2[i] = 1

y_1 = np.reshape(y_1, shape)
y_2 = np.reshape(y_2, shape)

print(amrr.__str__())
f, axarr = plt.subplots(1, 2, figsize = (15, 20))

axarr[0].imshow(y_1, cmap = 'gray')
axarr[0].title.set_text('Image not covered by S')

axarr[1].imshow(y_2, cmap = 'gray')
axarr[1].title.set_text('Image indexed by S -> X(,:)w')

plt.show()
```

```
Alternating Minimization for Robust Regression
```



Nonnegative matrix factorization

Consider the nonnegative matrix factorization problem. For this problem, we showed that the modified multiplicative updates algorithm satisfies the nonnegativity constraints at each iteration. Consider now the nonnegative sparse coding problem:

$$\begin{aligned} \text{minimize}_{W,H}\ F(W,H) &:= \frac{1}{2}\|WH - X\|_F^2 + \lambda\|H\|_1 \\ \text{subj. to } W_{ij} &\geq 0\ \forall i,j \\ H_{ij} &\geq 0\ \forall i,j, \end{aligned}$$

where

$$\|H\|_1 = \sum_{i,j} |H_{ij}|.$$

and $\lambda \geq 0$ is a parameter that controls the effect of the l1-norm. This problem is called sparse coding because the l1-norm forces a lot of weights in matrix H to become zero.

In [74]:
```python
# This piece of code is for loading data and visualizing
# the first 6 images in the dataset.

# Useful packages for loading the data and plotting
from numpy.random import RandomState
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_olivetti_faces

n_row, n_col = 2, 3
image_shape = (64, 64)
rng = RandomState(0)

# Useful function for plotting
def plot_gallery(title, images, n_col=n_col, n_row=n_row, cmap=plt.cm
.gray):
    plt.figure(figsize=(2. * n_col, 2.26 * n_row))
    plt.suptitle(title, size=16)
    for i, comp in enumerate(images):
        plt.subplot(n_row, n_col, i + 1)
        vmax = max(comp.max(), -comp.min())
        plt.imshow(comp.reshape(image_shape), cmap=cmap,
                   interpolation='nearest',
                   vmin=-vmax, vmax=vmax)
        plt.xticks(())
        plt.yticks(())
    plt.subplots_adjust(0.12, 0.05, 0.99, 0.75, 0.04, 0.)

# Load faces data
dataset = fetch_olivetti_faces(shuffle=True, random_state=rng)

# Store the vectorized images. Each image has dimensions 64 x 64.
faces = dataset.data

print("Dataset consists of %d faces" % len(faces))

plot_gallery("First 6 faces from the dataset", faces[:6], 6, 1)
```

```
Dataset consists of 400 faces
```

Question 3

Show that the modified multiplicative updates algorithm for the above nonnegative sparse coding problem also satisfies the nonnegative constraints at each iteration.

Marks: 5

See attached

Question 4

Use the face dataset, see Assigment 5. Set parameter $r = 6$ in the nonnegative factorization problem. Plot $\frac{1}{2}\|WH - X\|_F^2$ as λ increases.

Marks: 12

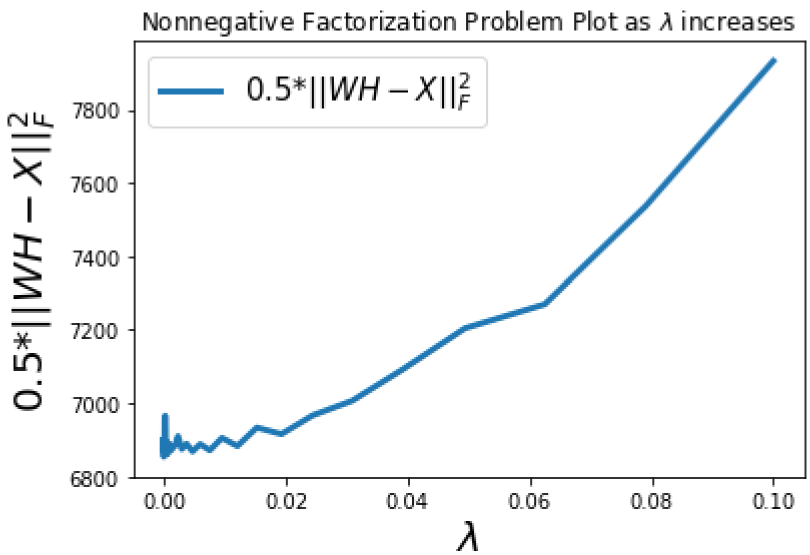
In [75]:
```
class NonNegMatFac():
    def __init__(self, lambda_, data):
        self.lambda_ = lambda_
        self.sigma = initParams['sigma']
        self.delta = initParams['delta']
        self.X = data.T

    def __str__(self):
        return 'Non-negative Matrix Factorization'

    def objFunc(self, W, H):
        X = self.X
        return 0.5 * np.linalg.norm(W @ H - X, ord = 'fro') ** 2 + se
lf.lambda_ * np.linalg.norm(H, 1)

    def gradH(self, W, H):
        X = self.X
        temp = W.T @ (W @ H - X)
        return temp + np.full(temp.shape, self.lambda_)

    def gradW(self, W, H):
        X = self.X
        return (W @ H - X) @ H.T

    def HBar(self, W, H):
        gradH = self.gradH(W, H)
        HBar = np.where(gradH >= 0, H, np.maximum(H, self.sigma * np.
ones((H.shape))))
        return HBar

    def WBar(self, W, H):
        gradW = self.gradW(W, H)
        WBar = np.where(gradW >= 0, W, np.maximum(W, self.sigma * np.
ones((W.shape))))
        return WBar

    def bars(self, W, H):
        HBar = self.HBar(W, H)
        WBar = self.WBar(W, H)
        return HBar, WBar

    def normalize(self, r, W, H):
        S = np.eye(r) / np.sum(W, 0)
        W = W @ S
        H = np.linalg.inv(S) @ H
        return W, H
```

In [76]:
```
def nonNegMatFac(lambda_, data, r = 6):
    m, n = data.T.shape
    max_iters = initParams['max_iter']
    nnmf = NonNegMatFac(lambda_, data)
    W = np.random.uniform(0, 1, size = (m, r))
    H = np.random.uniform(0, 1, size = (r, n))
    delta = initParams['delta']

    for i in range(max_iters):
        HBar = nnmf.HBar(W, H)
        H = H - np.divide(HBar, ((W.T @ W @ HBar) + delta)) * nnmf.gr
adH(W, H)

        WBar = nnmf.WBar(W, H)
        W = W - np.divide(WBar, ((WBar @ H @ HBar.T) + delta)) * nnmf
.gradW(W, H)

        #Normalize
        W, H = nnmf.normalize(r, W, H)
    return W, H
```

In [79]:
```
WHData = []
lambda_list = np.power(10, np.linspace(-6, -1, 50))
start = time.time()
for i in range(len(lambda_list)):
    if (i % 5 == 0):
        print('Lambda =', lambda_list[i])
        print('Starting number', i + 1, 'of', len(lambda_list))

    W, H = nonNegMatFac(lambda_list[i], faces)
    WHData.append(0.5 * np.linalg.norm(W @ H - faces.T, ord = 'fro')
** 2)

    if (i % 5 == 0):
        print('Computed W and H for lambda Value:', lambda_list[i])
        print()
print('Done computing W and H for all', len(lambda_list), 'lambdas i
n', time.time() - start, 's')
```

```
Lambda = 1e-06
Starting number 1 of 50
Computed W and H for lambda Value: 1e-06

Lambda = 3.2374575428176467e-06
Starting number 6 of 50
Computed W and H for lambda Value: 3.2374575428176467e-06

Lambda = 1.0481131341546853e-05
Starting number 11 of 50
Computed W and H for lambda Value: 1.0481131341546853e-05

Lambda = 3.39322177189533e-05
Starting number 16 of 50
Computed W and H for lambda Value: 3.39322177189533e-05

Lambda = 0.00010985411419875583
Starting number 21 of 50
Computed W and H for lambda Value: 0.00010985411419875583

Lambda = 0.00035564803062231287
Starting number 26 of 50
Computed W and H for lambda Value: 0.00035564803062231287

Lambda = 0.0011513953993264481
Starting number 31 of 50
Computed W and H for lambda Value: 0.0011513953993264481

Lambda = 0.0037275937203149418
Starting number 36 of 50
Computed W and H for lambda Value: 0.0037275937203149418

Lambda = 0.012067926406393288
Starting number 41 of 50
Computed W and H for lambda Value: 0.012067926406393288

Lambda = 0.03906939937054621
Starting number 46 of 50
Computed W and H for lambda Value: 0.03906939937054621

Done computing for all 50 lambdas in 3987.780258655548 s
```

In [92]:
```
plt.plot(lambda_list, WHData, label = r"0.5*$\||WH-X||^2_F$", linewid
th = 3)
plt.legend(fontsize = 15)
plt.xlabel("$\lambda$", fontsize = 20)
plt.ylabel(r"0.5*$\||WH-X||^2_F$", fontsize = 20)
# plt.xscale('log')
plt.title('Nonnegative Factorization Problem Plot as $\lambda$ increa
ses')
plt.show()
```



Question 5

Choose a λ and extract the features matrix W by solving the nonnegative matrix factorization problem. Report the 6 features of the faces dataset, i.e., the 6 columns of matrix W. You can report the features by visualizing them in a similar way to the above example.

Marks: 12

In [81]:
```
W, H = nonNegMatFac(initParams['lambda_'], faces)
plot_gallery("Features", W.T[:6], 6, 1)
```

Features