(and similarities) in what you found above?

d) [30 pts]

Write a python function that, given a matrix A, performs the QR algorithm applied to A^TA and returns U and $\Sigma^T\Sigma$. You may use Hessenberg form in Python (la.hessenberg) and QR decomposition (la.qr) in the scipy.linalg module. Type help(la.hessenberg) and help(la.qr) for more details. Note by itself, this is not the SVD decomposition yet, but is an important step in it. **No credit if you use the la.svd directly**!

In [ ]:

e) [10 pts]

As a modification, try to shift the QR algorithm. That is, instead of

$$H^{(0)} = Q_0R_0 \to H^{(1)} = R_0Q_0, \tag{2}$$

iterate this time on $H^{(0)} - \tau I$:

$$H^{(0)} - \tau I = Q_0R_0 \to H^{(1)} = R_0Q_0 + \tau I. \tag{3}$$

You may, for instance start with

$$\tau = H_{dd}^{(0)} \tag{4}$$

and change to $H_{d-1d-1}^{(k)}$ when the off diagonal elements of the dth row of $H^{(k)}$ ($H_{dd-1}^{(k)}$) are sufficiently small, and so on...

In [ ]:

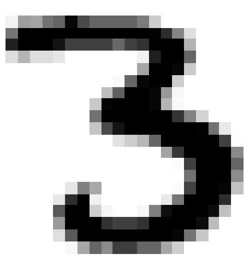
Image compression step

The following code snippet (in the cell below this one) loads the file image.npy as the matrix A (this is a pixel image of the digit 3).

In [4]:

```
# Code here imports the image and plots the original. Note it is
# a 28x28 numpy array

A = np.load('image.npy')
plt.imshow(A, cmap="binary")
plt.axis("off")
plt.show()
```



f) [30 pts]

Use the python function from d) or e), whichever you prefer, to find

$$\Sigma_1, U_1, V_1 \tag{4}$$

from the matrix A resulting from the file image.npy. Then for the following values of k, obtain the rank-k approximations:

$$A_k = \sum_{i=1}^{k} \sigma_i \mathbf{v}_i \mathbf{u}_i^T, \quad k = 4, 10 \text{ and } r \tag{5}$$

and plot the corresponding images for each value of k used.