Proximal Denoising

Henrique Miyamoto and Ilyas Hanine

1. The proximity operator is defined as

$$\text{prox}_f(y) := \arg\min_{x \in \mathcal{H}} f(x) + \frac{1}{2}\|x - y\|^2,$$

so solving the problem

$$\min_{x \in \mathbb{R}^N} \frac{1}{2}\|x - y\|^2 + f(Lx)$$

amounts to computing the proxixmity operator $\text{prox}_{f \circ L}(y)$. Since L is a linear operator in $\mathbb{R}^N$ and each $f_i \in \Gamma_0(\mathbb{R})$, then $f \circ L \in \Gamma_0(\mathbb{R})$, the set of proper, convex and lower semi-continuous functions.

2. The proximity operator of a function composition is calculated as

$$\begin{aligned}
\text{prox}_{f \circ L} &= \text{Id} - L^* \circ (\text{Id} - \text{prox}_f(y)) \circ L \\
&= \text{Id} - L^* \circ \text{Id} \circ L + L^* \text{prox}_f \circ L \\
&= L^* \text{prox}_f \circ L.
\end{aligned}$$

As L is orthogonal, $L^* = L^{-1}$ and

$$\text{prox}_{f \circ L}(y) = L^{-1} \text{prox}_f(Ly).$$

3. We apply an additive white Gaussian noise (AWGN) to the image `florence.jpg`, with mean zero and standard deviation 30.

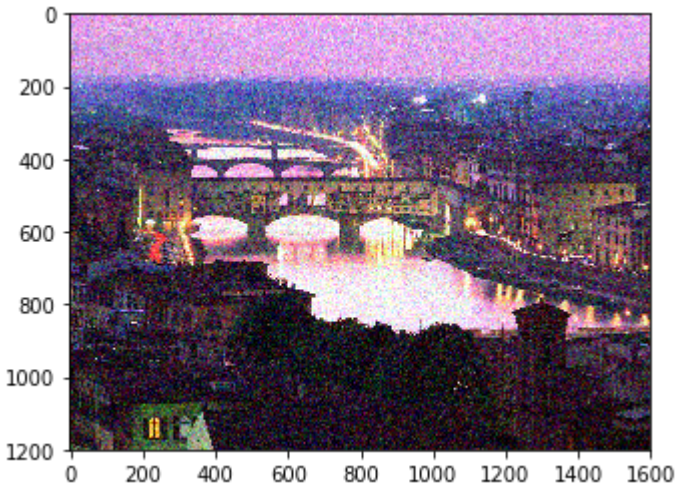
In [1]:
```
# Import packages
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
from fractions import Fraction
from decimal import Decimal
import pywt
```

In [2]:
```
# Import and show image
img = mpimg.imread("florence.jpg")
heigth = img.shape[0]
width = img.shape[1]
plt.imshow(img)
```

Out[2]: <matplotlib.image.AxesImage at 0x7f5c239c9c50>

In [3]:
```
# Add noise
noise = np.random.normal(0, 30, (heigth,width,3))
noisy = img + noise
noisy = np.clip(noisy, 0, 255)
noisy = noisy.astype(np.uint8)
plt.imshow(noisy)
plt.imsave("noisy.jpg", noisy)

MSE_noisy = np.linalg.norm(noisy - img)/np.linalg.norm(img)
print(MSE_noisy)
```

1.25941181084



4. We use the proximity operator approach to solve the minimisation problem when the image is decomposed in wavelets. We consider the function

$$f_i = \begin{cases} 0, & i \in \mathbb{K}, \\ \varphi, & \text{sinon}, \end{cases} \quad i = 1, \ldots, N,$$

where $\mathbb{K}$ is the index set of approximation coefficients and $\varphi = \chi|\cdot|^q$, with $q \in \{1, 4/3, 3/2, 2, 3, 4\}$and $\chi \in]0, +\infty[$.

We consider decomposition with two types of wavelets: Daubechies (`'db8'`) and Haar (`'haar'`) [1].

The table below presents the proximity operator for the relevant functions [2].

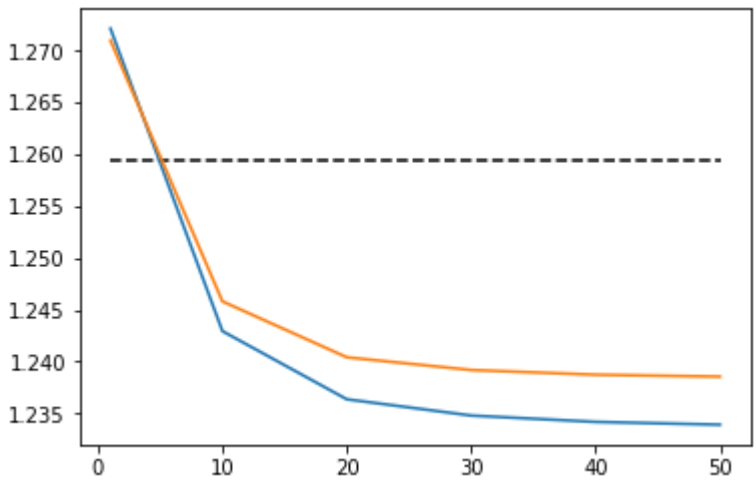
f	$\text{prox}_f(\xi)$
0	ξ
$\chi \cdot ^1$	$\mathrm{sign}(\xi)\max\{ \xi -\chi,0\}$
$\chi \cdot ^{4/3}$	$\xi + \frac{4\chi}{(3)(2^{1/3})}\left((\epsilon - \xi)^{1/3} - (\epsilon + \xi)^{1/3}\right),$ $\epsilon = \sqrt{\xi^2 + 256\chi^3/729}$
$\chi \cdot ^{3/2}$	$\xi+\frac{9\chi^2\mathrm{sign}(\xi)}{8}\left(1- \sqrt{1+\frac{16 \xi }{9\chi^2}} \right)$
$\chi \cdot ^2$	$\frac{\xi}{1+2\chi}$
$\chi \cdot ^3$	$\mathrm{sign}(\xi)\frac{\sqrt{1+12\chi \xi }-1}{6\chi}$
$\chi \cdot ^4$	$\left(\frac{\epsilon+\xi}{8\chi}\right)^{1/3} - \left(\frac{\epsilon-\xi}{8\chi}\right)^{1/3}, \quad \epsilon = \sqrt{\xi^2 + 1/(27\chi)}$

In [4]:

```
# Define prox function
# Input: xi (argument of the operator), q, chi (parameters)
# Output: result of prox operation
def prox(xi, q, chi):
    if q == 1:
        return np.sign(xi) * max(abs(xi)-chi, 0)
    if q == 2:
        return xi/(1+2*chi)
    if q == 3:
        return np.sign(xi)*(np.sqrt(1+12*chi*abs(xi))-
1)/(6*chi)
    if q == 4:
        epsilon = np.sqrt(xi**2 + 1/(27*chi))
        return ((epsilon+xi)/(8*chi))**(1/3) - ((epsilon-x
i)/(8*chi))**(1/3)
    if q == Fraction(4,3):
        epsilon = np.sqrt(xi**2 + (256*(chi**3)/729))
        return xi + (4*chi)*((epsilon-xi)**(1/3)-(epsilon+x
i)**(1/3))/(3*2**(1/3))
    if q == Fraction(3,2):
        return xi + (9*(chi**2)*np.sign(chi))*(1-np.sqrt
(1+(16*abs(xi)/(9*chi**2))))/(8)

# Define denoise function
# Input: noisy (noisy image), q, chi (parameters for prox c
alculation), wt_type (wavelet type)
# Output: proxfLy (denoised image)
def denoise(noisy, q, chi, wt_type):
    # Compute Ly
    coeffs = pywt.wavedec2(noisy, wt_type, level=1, axe
s=(0, 1))
    cA = coeffs[0];
    cH = coeffs[1][0];
    cV = coeffs[1][1];
    cD = coeffs[1][2];

    # Compute prox_f(Ly)
    proxcA = np.zeros((cA.shape[0], cA.shape[1], cA.shape
[2]))
    proxcH = np.zeros((cH.shape[0], cH.shape[1], cH.shape
[2]))
    proxcV = np.zeros((cV.shape[0], cV.shape[1], cV.shape
[2]))
    proxcD = np.zeros((cD.shape[0], cD.shape[1], cD.shape
[2]))

    # First quadrant
    proxcA = cA;

    # Second quadrant (horizontal coefficients)
    for i in range(cH.shape[0]):
        for j in range(cH.shape[1]):
            for k in range(cH.shape[2]):
                xi = cH[i,j,k]
                proxcH[i,j,k] = prox(xi,q,chi)
```

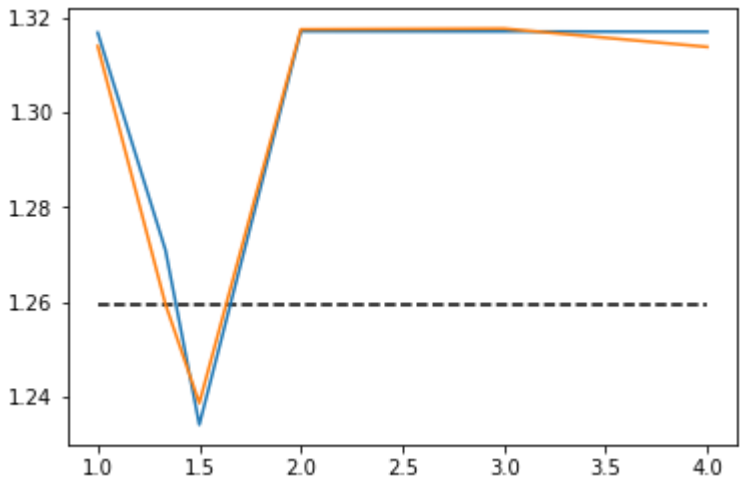
In [5]:
```
# For a fixed q, vary chi
q = Fraction(3,2);
chi_list = [1, 10, 20, 30, 40, 50]
wt_list = ['db8', 'haar']

for wt in wt_list:
    MSE_list = []
    for chi in chi_list:
        img_hat = denoise(noisy, q, chi, wt)
        MSE = np.linalg.norm(img_hat - img)/np.linalg.norm
(img)
        MSE_list.append(MSE)
    plt.plot(chi_list, MSE_list)
plt.hlines(MSE_noisy, 1, 50, linestyles='dashed')
```

Out[5]:
```
<matplotlib.collections.LineCollection at 0x7f5c2395a8d0>
```



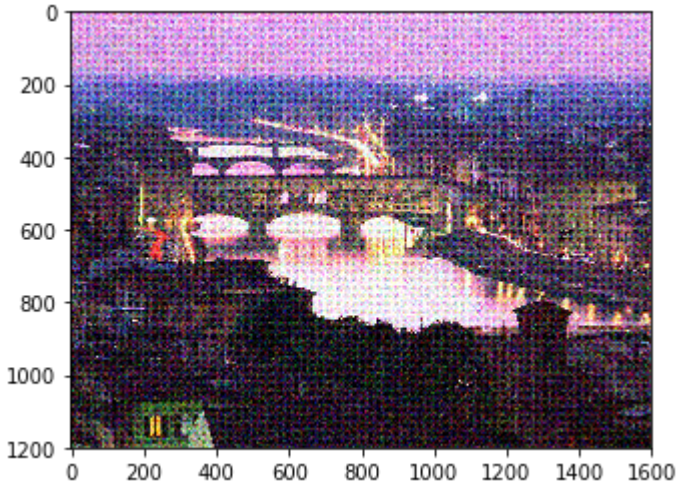
In [6]:
```
# For a fixed chi, vary q
chi = 40
q_list = [1, Fraction(4,3), Fraction(3,2), 2, 3, 4]
wt_list = ['db8', 'haar']

for wt in wt_list:
    MSE_list = []
    for q in q_list:
        img_hat = denoise(noisy, q, chi, wt)
        MSE = np.linalg.norm(img_hat - img)/np.linalg.norm
(img)
        MSE_list.append(MSE)
    plt.plot(q_list, MSE_list)
plt.hlines(MSE_noisy, 1, 4, linestyles='dashed')
```

Out[6]:
```
<matplotlib.collections.LineCollection at 0x7f5c2395ad10>
```



In [7]:
```
# Display result for best parameters
q = Fraction(3,2)
chi = 50
wt = 'db8'

img_hat = denoise(noisy, q, chi, wt)
plt.imshow(img_hat)
plt.imsave("result.jpg", img_hat)

MSE = np.linalg.norm(img_hat - img)/np.linalg.norm(img)
print(MSE)
```

```
1.23389060663
```



References

[1] Gregory R. Lee, Ralf Gommers, Filip Wasilewski, Kai Wohlfahrt, Aaron O'Leary (2019). *PyWavelets: A Python package for wavelet analysis*. Journal of Open Source Software, 4(36), 1237, https://doi.org/10.21105/joss.01237 (https://doi.org/10.21105/joss.01237).

[2] Emilie Chouzenoux and Jean-Christophe Pesquet. *Large Scale and Distributed Optimization*, Part IV: Proximity operator. Course notes, CentraleSupélec, 2019.