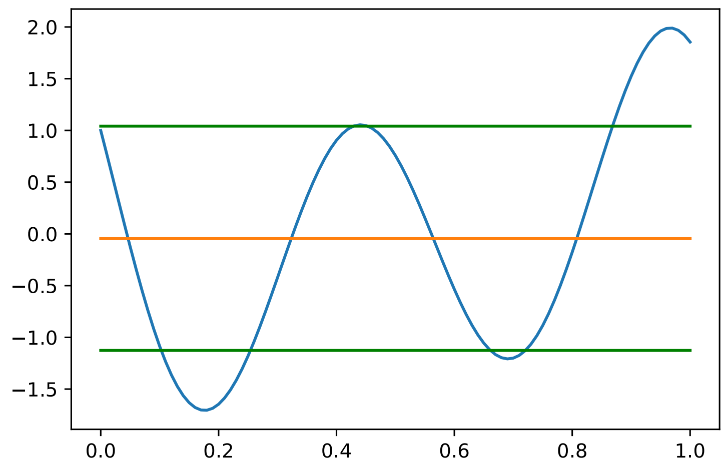

Figure 3.1.: A signal x. The horizontal lines show $\mathbf{avg}(x) + \mathbf{rms}(x)$, $\mathbf{avg}(x)$, and $\mathbf{avg}(x) - \mathbf{rms}(x)$.

```
79
```

In [ ]:
```
# Number of entries of x with |x_i| >= a
print(sum(abs(x) >= a))
```

```
20
```

In the last line, the expression `abs(x) >= a` creates an array with entries that are Boolean, *i.e.*, `true` or `false`, depending on whether the corresponding entry of `x` satisfies the inequality. When we sum the vector of Boolean, they are automatically converted to the numbers 1 and 0, respectively.

3.2. Distance

Distance. The distance between two vectors is $\mathbf{dist}(x, y) = \|x - y\|$. This is written in Python as `np.linalg.norm(x-y)`. Let's find the distance between the pairs of the three vectors u, v, and w from page 49 of VMLS.

In [ ]:
```
u = np.array([1.8, 2.0, -3.7, 4.7])
v = np.array([0.6, 2.1, 1.9, -1.4])
w = np.array([2.0, 1.9, -4.0, 4.6])
```

```
print(np.linalg.norm(u-v))
print(np.linalg.norm(u-w))
print(np.linalg.norm(v-w))
```

```
8.367795408588812
0.3872983346207417
8.532877591996735
```

We can see that u and w are much closer to each other than u and v, or v and w. Other expressions such as `np.sqrt(sum((a-b)**2))` and `sum((a-b)**2)**0.5` also give the distance between vector a and b.

Nearest neighbor. We define a function that calculates the nearest neighbour of a vector in a list of vectors, and try it on the points in Figure 3.3 of VMLS.

```
In [ ]: near_neigh = lambda x,z: z [np.argmin([np.linalg.norm(x-y) for y
        ↪ in z])]
        z = ([2,1], [7,2], [5.5,4], [4,8], [1,5], [9,6])
        x = np.array([5,6])
        print(near_neigh(x,z))
```

```
[5.5, 4]
```

```
In [ ]: print(near_neigh(np.array([3,3]),z))
```

```
[2, 1]
```

On the first line, the expression `[np.linalg.norm(x-y)for y in z]` uses a convenient construction in Python. Here `z` is a list of vectors, and the expression expands to an array with elements `np.linalg.norm(x-z[0]), np.linalg.norm(x-z[1]),...` The numpy function `np.argmin` applied to this array returns the index of the smallest element.

De-meaning a vector. We refer to the vector $x - \mathbf{avg}(x)\mathbf{1}$ as the de-meaned version of x.

```
In [ ]: de_mean = lambda x: x - sum(x)/len(x)
        x = np.array([1,-2.2,3])
        print ('Average of x: ', np.mean(x))
        x_tilde = de_mean(x)
        print('x_tilde: ',x_tilde)
        print('Average of x_tilde: ',np.mean(x_tilde))
```

```
Average of x:  0.6
x_tilde:  [ 0.4 -2.8  2.4]
Average of x_tilde:  -1.4802973661668753e-16
```

(The mean of $\tilde{x}$ is very very close to zero.)

3.3. Standard deviation

Standard deviation. We can define a function that corresponding to the VMLS definition of the standard deviation of a vector, $\mathbf{std}(x) = \|x - \mathbf{avg}(x)\mathbf{1}\|/\sqrt{n}$, where n is the length of the vector.

In [ ]:
```
x = np.random.random(100)
stdev = lambda x: np.linalg.norm(x - sum(x)/len(x))/(len(x)**0.5)
stdev(x)
```

Out[ ]:
```
0.30440692170248823
```

You can also use the `numpy` function `np.std(x)` to obtain the standard deviation of a vector.

Return and risk. We evaluate the mean return and risk (measured by standard deviation) of the four time series Figure 3.4 of VMLS.

In [ ]:
```
a = np.ones(10)
np.mean(a), np.std(a)
```

Out[ ]:
```
(1.0, 0.0)
```

In [ ]:
```
b = [ 5, 1, -2, 3, 6, 3, -1, 3, 4, 1 ]
np.mean(b), np.std(b)
```

Out[ ]:
```
(2.3, 2.4103941586387903)
```

In [ ]:
```
c = [ 5, 7, -2, 2, -3, 1, -1, 2, 7, 8 ]
np.mean(c), np.std(c)
```

Out[ ]:
```
(2.6, 3.7735924528226414)
```

In [ ]:
```
d = [ -1, -3, -4, -3, 7, -1, 0, 3, 9, 5 ]
np.mean(d), np.std(d)
```

Out[ ]:
```
(1.2, 4.308131845707604)
```