Monotonce Cubic interpolation

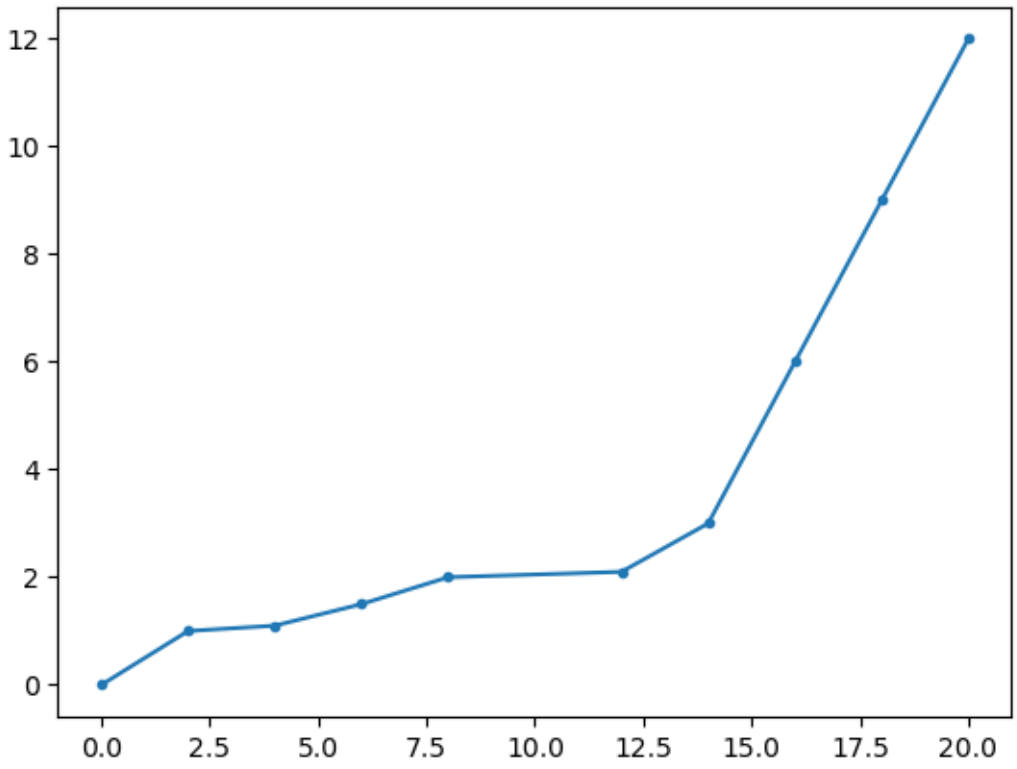
January 5, 2023

https://en.wikipedia.org/wiki/Monotone_cubic_interpolation

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[3]:
```
xs = np.asarray([0, 1, 2, 3, 4, 6, 7, 8, 9, 10]) * 2
ys = np.asarray([0, 1, 1.1, 1.5, 2, 2.1, 3, 6, 9, 12])

plt.plot(xs, ys, '.-')
```

[3]:
```
[<matplotlib.lines.Line2D at 0x7fc2d3a9a800>]
```



[4]:
```
# 1
#d = np.diff(xs) / np.diff(ys)
#d = np.gradient(xs) / np.gradient(ys)
```

[5]:
```
# 2
#m = np.diff(d) / 2
#m = np.gradient(d) / 2
```

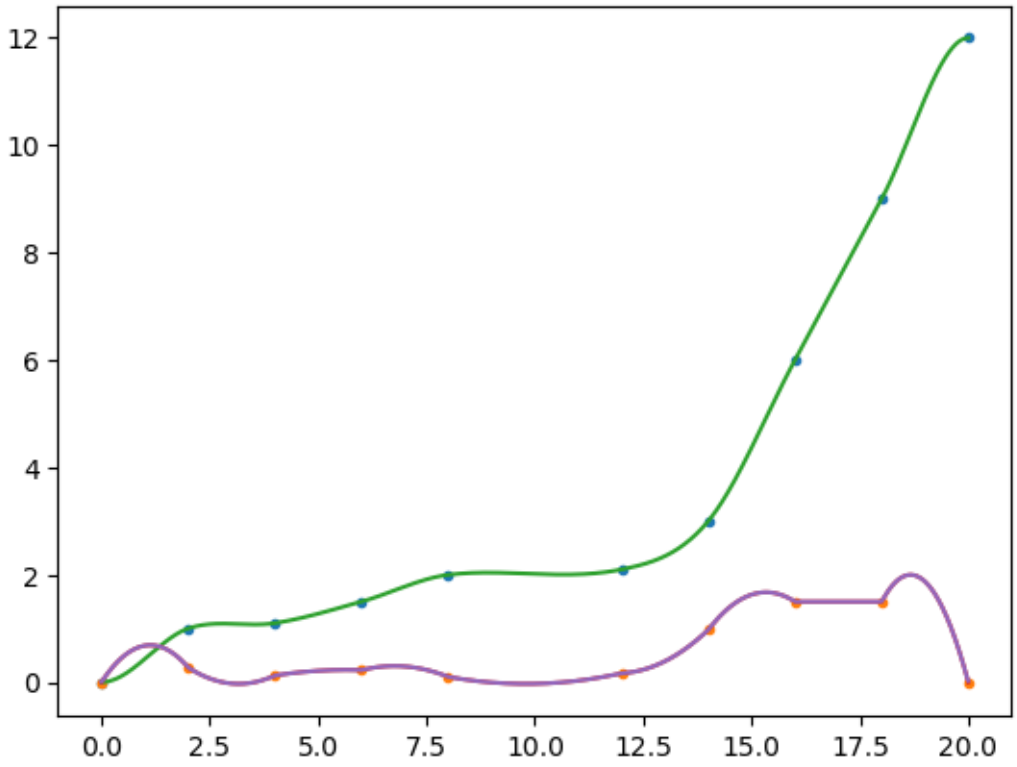
[6]:
```
# 4
#alpha = m/d
```

[14]:
```
m = np.gradient(ys) /  np.gradient(xs)
# force slope 0 at start and end
m[0] = 0
m[-1] = 0

# Cubic interp
x = np.linspace(1e-6, 20, 1000)
xk = np.searchsorted(xs, x) - 1
DELTA = np.diff(xs)  # Abstand der Stuetzstellen
t = (x - xs[xk]) / DELTA[xk]

def h00(t):
    return 2*t**3 - 3*t**2 + 1
def h10(t):
    return t**3 - 2*t**2 + t
def h01(t):
    return -2*t**3 + 3*t**2
def h11(t):
    return t**3 - t**2

# derivatives
def h00d(t):
    return 6*t**2 - 6*t
def h10d(t):
    return 3*t**2 - 4*t + 1
def h01d(t):
    return -6*t**2 + 6*t
def h11d(t):
    return 3*t**2 - 2*t

y = ys[xk] * h00(t) \
    + ys[xk+1] * h01(t) \
    + DELTA[xk] * m[xk+1] * h11(t) \
    + DELTA[xk] * m[xk] * h10(t)

yd = ys[xk] / DELTA[xk] * h00d(t) \
```

```
    + ys[xk+1] / DELTA[xk] * h01d(t) \
    + m[xk+1] * h11d(t) \
    + m[xk] * h10d(t)
```

[15]:
```
plt.plot(xs, ys, '.', xs, m, '.')
plt.plot(x, y)
plt.plot(x, yd)
plt.plot(x, np.gradient(y)/np.gradient(x))
```

[15]: [<matplotlib.lines.Line2D at 0x7fc2cb61b1f0>]



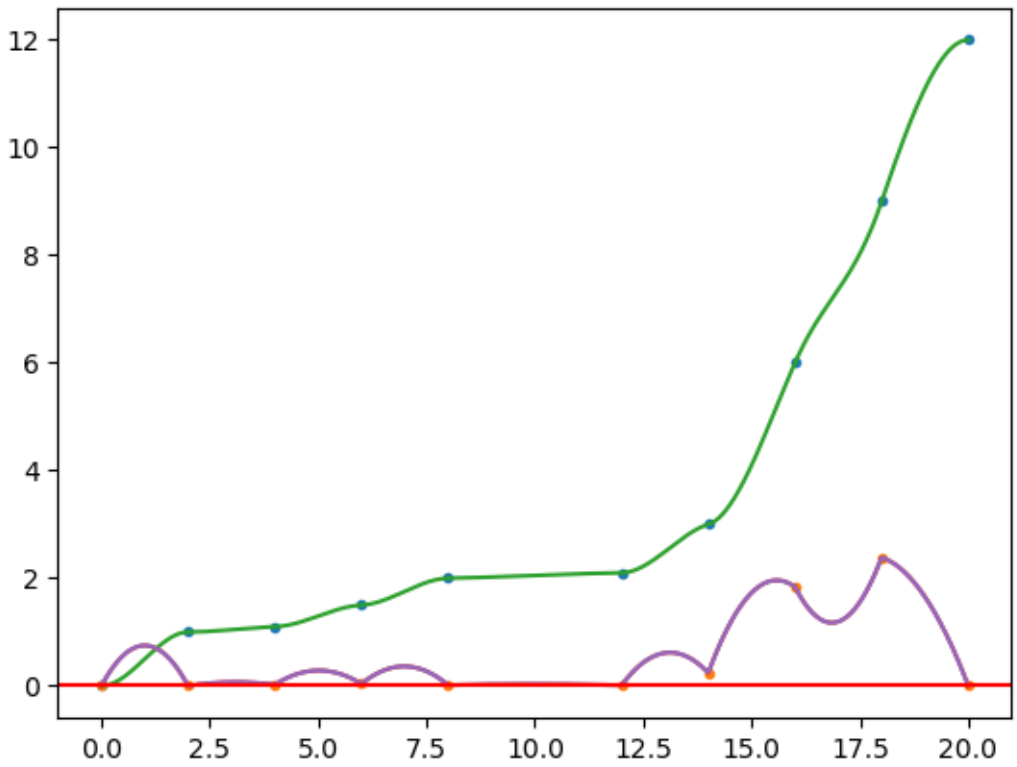
1 Monotonic

[71]:
```
m = np.gradient(ys) /  np.gradient(xs)
# force slope 0 at start and end
m[0] = 0
m[-1] = 0

# Cubic interp
x = np.linspace(1e-6, 20, 1000)
xk = np.searchsorted(xs, x) - 1
```

```
DELTA = np.diff(xs)  # Abstand der Stuetzstellen
t = (x - xs[xk]) / DELTA[xk]

def h00(t):
    return 2*t**3 - 3*t**2 + 1
def h10(t):
    return t**3 - 2*t**2 + t
def h01(t):
    return -2*t**3 + 3*t**2
def h11(t):
    return t**3 - t**2

# derivatives
def h00d(t):
    return 6*t**2 - 6*t
def h10d(t):
    return 3*t**2 - 4*t + 1
def h01d(t):
    return -6*t**2 + 6*t
def h11d(t):
    return 3*t**2 - 2*t

alpha = m[:-1] / DELTA
beta = m[1:] / DELTA
tau = 3 / np.sqrt(alpha**2 + beta**2)
m[1:] = tau * alpha * beta * m[1:]

y = ys[xk] * h00(t) \
    + ys[xk+1] * h01(t) \
    + DELTA[xk] * m[xk+1] * h11(t) \
    + DELTA[xk] * m[xk] * h10(t)

yd = ys[xk] / DELTA[xk] * h00d(t) \
    + ys[xk+1] / DELTA[xk] * h01d(t) \
    + m[xk+1] * h11d(t) \
    + m[xk] * h10d(t)
```

[72]:
```
plt.plot(xs, ys, '.', xs, m, '.')
plt.plot(x, y)
plt.plot(x, yd)
#plt.plot(xs[1:], alpha, '.', xs[1:], beta, '.', xs[1:], tau, '.')
plt.plot(x, np.gradient(y)/np.gradient(x))
plt.axhline(0, c='r')
```

[72]:
```
<matplotlib.lines.Line2D at 0x7fc2c93e8ee0>
```



2 Akima aus scipy

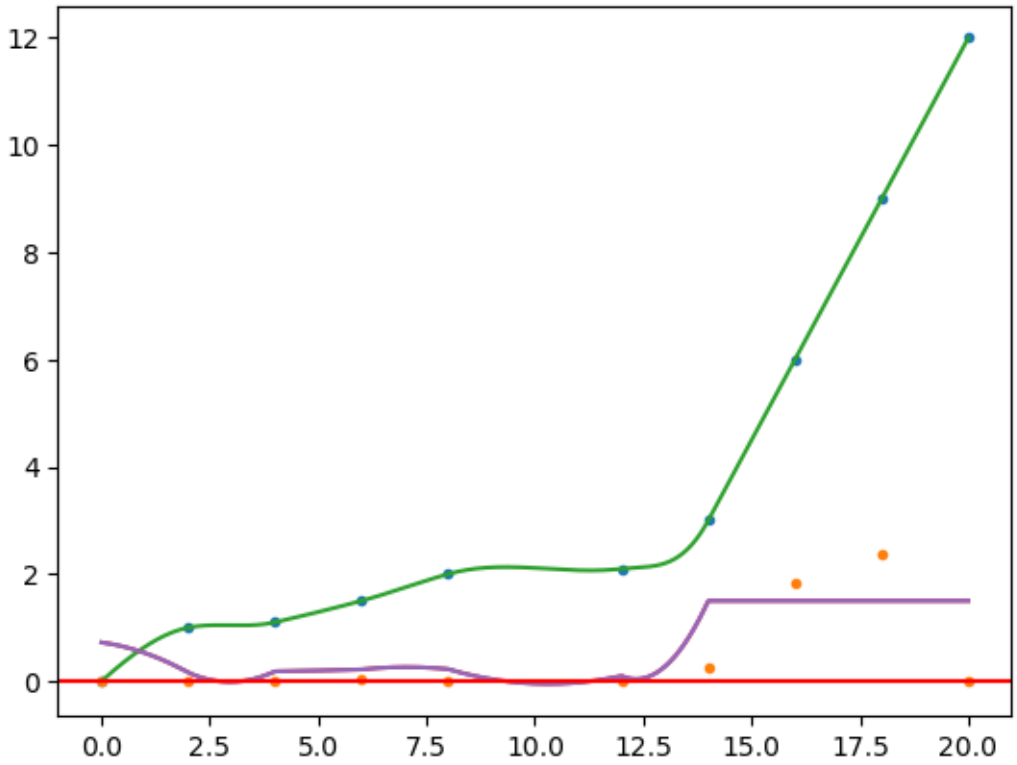
https://de.wikipedia.org/wiki/Akima-Interpolation

[73]:
```
from scipy.interpolate import Akima1DInterpolator
```

[96]:
```
f = Akima1DInterpolator(xs, ys)

x = np.linspace(0, 20, 1000)
y = f(x)
yd = f.derivative()(x)
```

[100]:
```
plt.plot(xs, ys, '.', xs, m, '.')
plt.plot(x, y)
plt.plot(x, yd)
#plt.plot(xs[1:], alpha, '.', xs[1:], beta, '.', xs[1:], tau, '.')
plt.plot(x, np.gradient(y)/np.gradient(x))
plt.axhline(0, c='r')
```

[100]:
```
<matplotlib.lines.Line2D at 0x7fc2c63cd870>
```



[102]:
```
np.gradient(ys)/np.gradient(xs)
```

[102]:
```
array([0.5       , 0.275     , 0.125     , 0.225     , 0.1       ,
       0.16666667, 0.975     , 1.5       , 1.5       , 1.5       ])
```

[ ]:
```

```