Oblig 2 MAT160

Sigbjørn Fjelland

October 22, 2021

1 Oppgave

Benyttet linspace funksjon i numpy til å generere 9 jevnt fordelte punkt:

```
xi = np.linspace(start = -1.0, stop=1.0, num= 9, endpoint= True)
```

og fikk følgende output:

i	1	2	3	4	5	6	7	8	9
x_i	-1	-0.75	-0.5	-0.25	0	0.25	0.5	0.75	1
$R(x_i)$	0.0385	0.0664	0.1379	0.3902	1	0.3902	0.1379	0.0664	0.0385

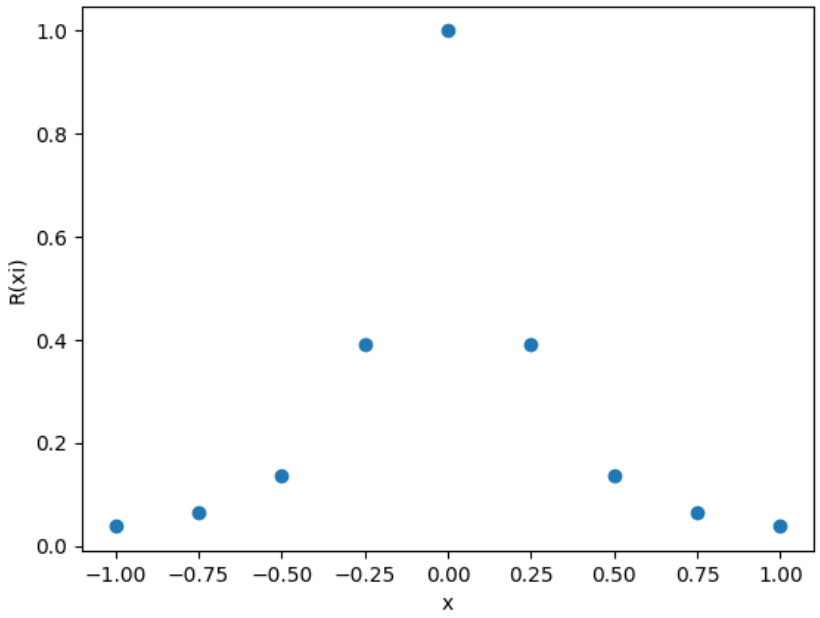

Figure 1: Interpolation points.

2 Oppgave

Graden av ett Lagrange polynom er definert ut fra hvor mange datapunkt vi har. Ved n målepunkt har man $n-1$. Vi skal ha 8 grads polynom og må dermed bruke alle målepunktene.

$$P_{n-1}(x_k) = \sum_{i=0}^{n} y_i \cdot l_i \tag{1}$$

$$l_i(x) = \prod_{i \neq j}^{n} \frac{x - x_j}{x_i - x_j} \tag{2}$$

utregning (2) er forsøkt kodet inn i python ved å loope to ganger over over de n antall målingnene for å lage en indeks for i og j og mellomlagre den i en ny l_i for alle x på linje 2 i code chunken. Den nye l_i blir i i sin tur blir lagt i en liste utenfor loopene.

```
for i in range(n):  # Loop over and li
li = np.ones(len(x))
for j in range(n):  # Loop to make the product for li
    if i is not j:
        li = li * (x - xi[j]) / (xi[i] - xi[j])
    l.append(li)  # Append the array to the list
```

Siste del tar seg av summeringen i utregning (1) ved å loope over n nok en gang:

```
P = 0
for k in range(len(yi)):
    P = P + yi[k] * l[k]
```

resultatet er det noe volatile plottet under

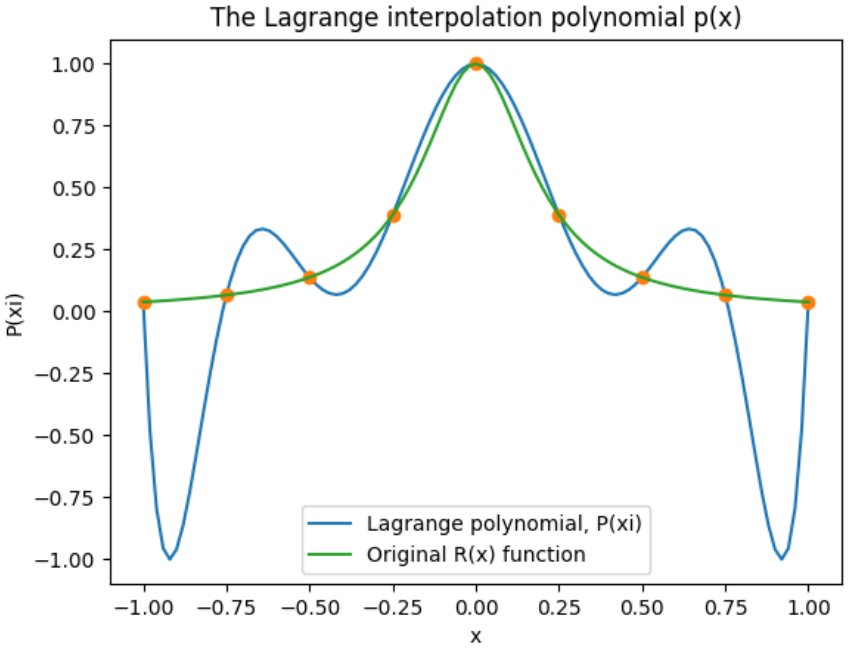

Figure 2: Lagrange interpolation.

3 Oppgave

For implementasjonen (også i python) er fremgangsmåten er som følger delt opp i chunks:

beregner $\Delta_i = x_{i+1} - x_i$ og $\delta_i = y_{i+1} - y_i$:

```
n = len(xi)
delta_x = np.diff(xi)
delta_y = np.diff(yi)
```

Generer A matrisen ut fra δ og b- vektor ut fra Δ: ved følgende ledd i koden:

```
### Get matrix A
    A = np.zeros(shape=(n, n))
    b = np.zeros(shape=(n, 1))
    A[0, 0] = 1
    A[-1, -1] = 1

    for i in range(1, n - 1):
        A[i, i - 1] = delta_x[i - 1]
        A[i, i + 1] = delta_x[i]
        A[i, i] = 2 * (delta_x[i - 1] + delta_x[i])
        ### Get matrix b
        b[i, 0] = 3 * (delta_y[i] / delta_x[i] - delta_y[i - 1] / delta_x[i - 1])
```

og skal få ut omtrentlig følgende A matrise og b vektor:

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0.25 & 1 & 0.25 & \ddots & & & & \vdots \\ 0 & 0.25 & 1 & 0.25 & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & 0.25 & 1 & 0.25 & 0 \\ \vdots & & & & \ddots & 0.25 & 1 & 0.25 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$b = \begin{bmatrix} 0 \\ 0.5233 \\ 2.1692 \\ 4.2893 \\ -14.6341 \\ 4.2893 \\ 2.1692 \\ 0.5233 \\ 0 \end{bmatrix} \tag{4}$$

Videre løser vi c i $Ac = b$ med numpy.linalgs solve, for videre å løse c og d ved code chunk:

```
### Solves for d and b
d = np.zeros(shape=(n - 1, 1))
b = np.zeros(shape=(n - 1, 1))
for i in range(0, len(d)):
    d[i] = (c[i + 1] - c[i]) / (3 * delta_x[i])
    b[i] = (delta_y[i] / delta_x[i]) - (delta_x[i] / 3) * (2 * c[i] + c[i + 1])
```

I siste instans beregnes hver spline og lagres i en felles liste av splines. Plottet blir satt sammen av flere subplot av splines på sitt respektive intervall, og orginalplot som benchmark:

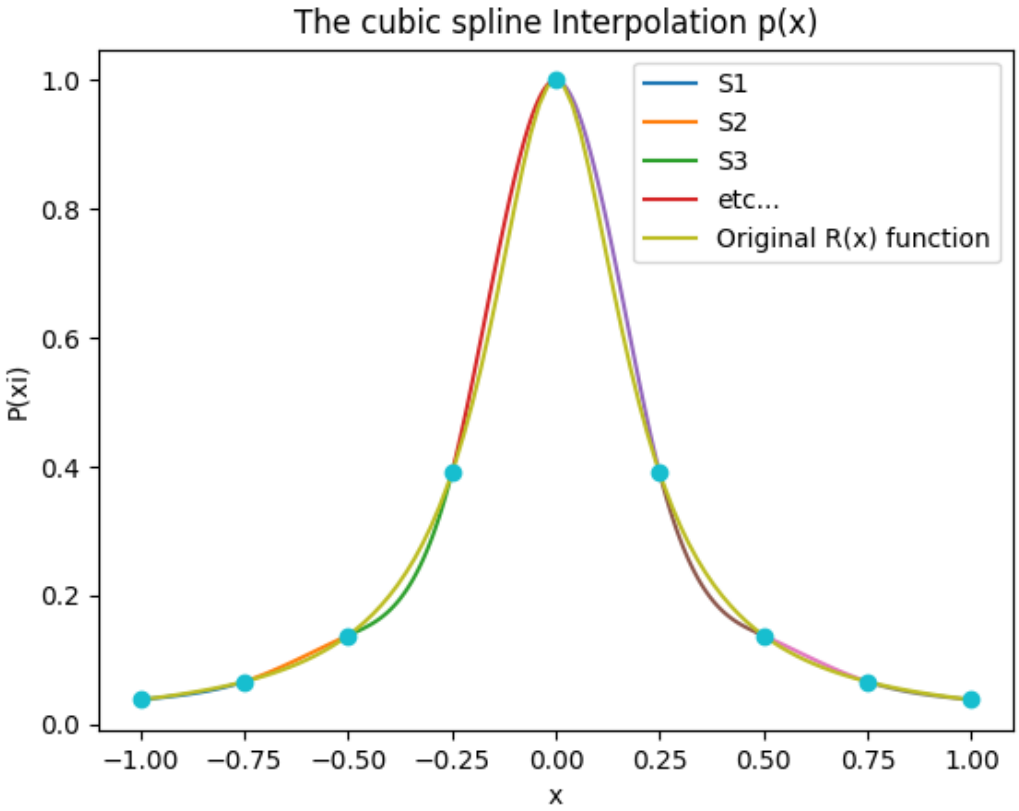

Figure 3: Natural cubic Spline.

4 Oppgave

Som vi ser direkte av kurven øker avvikene på Lagrange polynomene jo lengre ut mot randsonene en kommer, noe som skyldes runges fenomen. Til sammenligning følger splinsene med mye mindre svingninger om kurven.

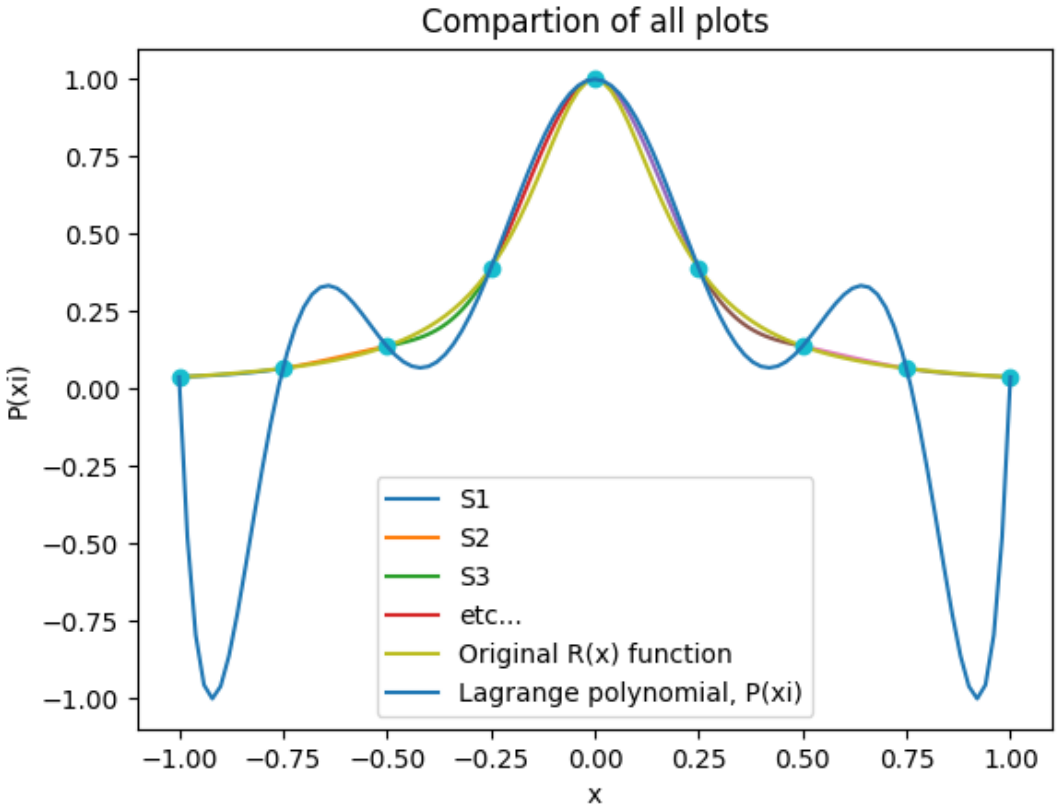

Figure 4: Sammenligning av alle plot