In [1]:
```
import time
import numpy as np
import scipy.interpolate as ip
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.pyplot import figure
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
```

In [3]:
```
SIZE = 5
MAX = 100
```

In [205]:
```
x = np.random.rand(SIZE) * MAX
y = np.random.rand(SIZE) * MAX
display(x, y)
```

```
array([61.74137569, 40.95997402, 22.74151601, 82.85679589, 45.83718844])
```

```
array([24.63874464, 83.74209807,  2.71344724, 53.61043903, 48.40252907])
```

In [5]:
```
def e2latex(e):
    if(e.find('e') == -1):
        return e
    [a, b] = e.split('e')
    return f'({a}\cdot 10^{{{b}}})'

def poly2latex(p):
    terms = []
    if len(p) > 2:
        for i in range(2, len(p)):
            ii = len(p) - i + 1
            term = 'x^{%d}' % ii
            c = p.coef[ii]
            if c!=1:
                cc = '%.2g ' % c
                term = e2latex(cc) + term
            terms.append(term)
    if len(p) > 1:
        term = 'x'
        c = p.coef[1]
        if c!=1:
            cc = '%.2g ' % c
            term = e2latex(cc) + term
        terms.append(term)
    cc = '%.2g' % p.coef[0]
    terms.append(e2latex(cc))
    px = '$P(x)=%s$' % '+'.join(terms)
    px = px.replace('+-', '-')
    return px

def display_poly(poly):
    p = np.polynomial.Polynomial(poly.c[::-1])
    display(Latex(poly2latex(p)))

def plot_poly(pd, xs, ys):
    plot_polys([pd], xs, ys)

def plot_polys(plist, xs, ys):
    t = np.linspace(0, MAX, 500)
    figure(num=None, figsize=(11, 8), dpi=80, facecolor='w', edgecolor='k')
    for (p, desc) in plist:
        plt.plot(t, p(t), label=desc)
    plt.scatter(xs, ys, label="Sample")
    plt.legend(bbox_to_anchor=(0., -.200, 1., .200), loc=10, ncol=5)
    plt.xlabel('$x$')
    plt.ylabel('$P(x)$')
```

Interplacja Lagrenge'a

Wielomian interpolacyjny Lagrange'a:

$$P_n(x) = \sum_{j=1}^{n} \left(y_j \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} \right)$$

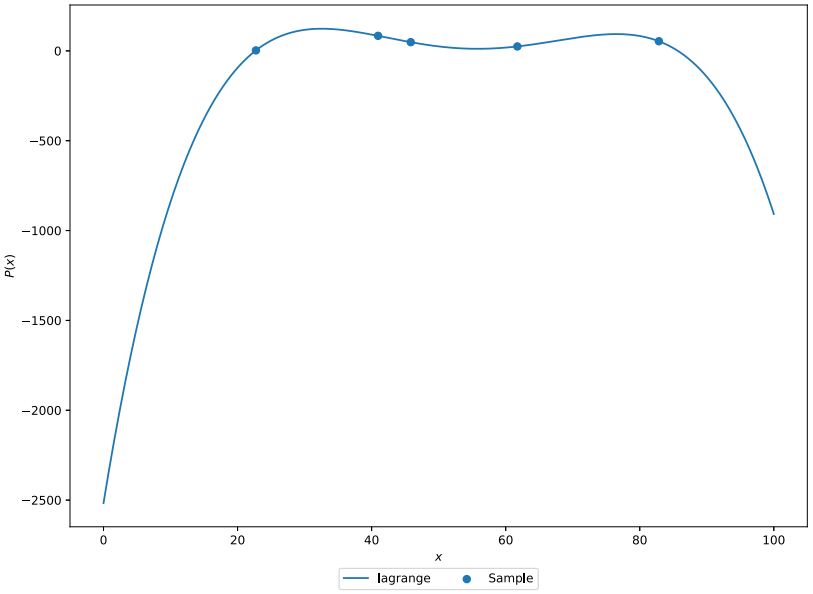
In [230]:
```
def my_lagrange(x, y):
    assert x.size == y.size
    result = np.poly1d(0)
    for j in range(x.size):
        result += my_lagrange_Pj(x, y, j)
    return result

def my_lagrange_Pj(x, y, j):
    p = np.poly1d([1,0])
    d = np.poly1d(1)
    m = 1.0
    for k in range(x.size):
        if(k == j):
            continue
        d *= p - x[k]
        m *= x[j] - x[k]
    return y[j]*(d/m)
```

In [208]:
```
p_my_lagrange = my_lagrange(x, y)
p_lagrange = ip.lagrange(x, y)

display_poly(p_lagrange)
display_poly(p_my_lagrange)
```

$P(x) = -0.00041x^4 + 0.091x^3 - 7.1x^2 + (2.3 \cdot 10^{+02})x + (-2.5 \cdot 10^{+03})$

$P(x) = -0.00041x^4 + 0.091x^3 - 7.1x^2 + (2.3 \cdot 10^{+02})x + (-2.5 \cdot 10^{+03})$

In [669]:
```
plot_poly((p_my_lagrange, "lagrange"), x, y)
```



Interpolacja Newtona

Wielomian interpolacyjny Newtona:

$$P_n(x) = \sum_{i=0}^{n} \left(a_i \prod_{j=0}^{i-1} (x - x_j) \right)$$

gdzie:

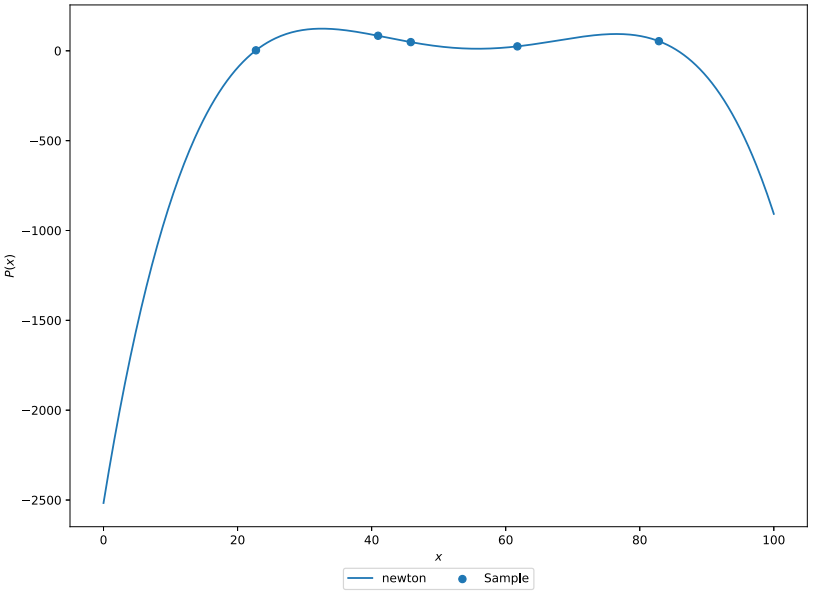
$$a_i = f[x_0, x_1, \ldots, x_i]$$

In [8]:
```
def newton_coef(x, y):
    n = x.size
    a = np.empty(n)
    for i in range(n):
        a[i] = y[i]
    for j in range(1, n):
        for i in range(n-1, j-1, -1):
            a[i] = (a[i] - a[i-1])/(x[i] - x[i-j])
    return a

def my_newton(x, y):
    assert x.size == y.size
    n = x.size
    a = newton_coef(x, y)
    result = np.poly1d(0)
    for i in range(n):
        p = np.poly1d(1)
        for j in range(i):
            p *= np.poly1d(x[j], True)
        result += a[i] * p
    return result
```

In [409]:
```
p_my_newton = my_newton(x, y)

display_poly(p_my_newton)
```

$P(x) = -0.00041x^4 + 0.091x^3 - 7.1x^2 + (2.3 \cdot 10^{+02})x + (-2.5 \cdot 10^{+03})$

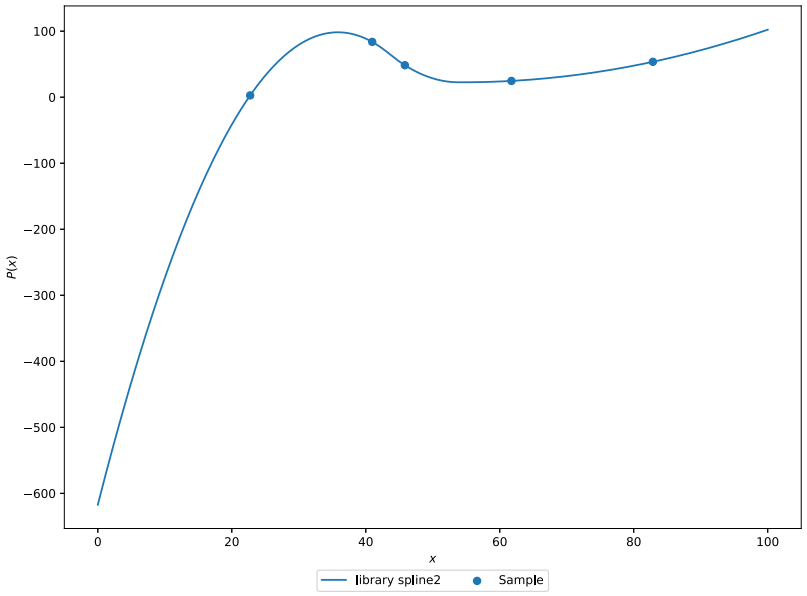
In [670]:
```
plot_poly((p_my_newton, "newton"), x, y)
```



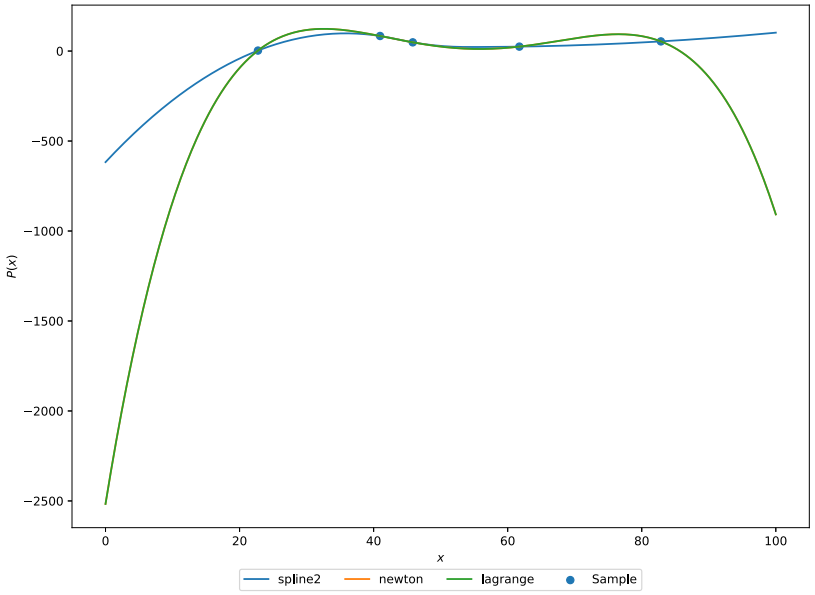
Biblioteczna implementacja interpolacji

(za pomocą krzywych sklejanych drugiego stopnia)

In [671]:
```
p_lib_interp = ip.interp1d(x, y, fill_value="extrapolate", kind="quadratic")

plot_poly((p_lib_interp, "library spline2"), x, y)
```



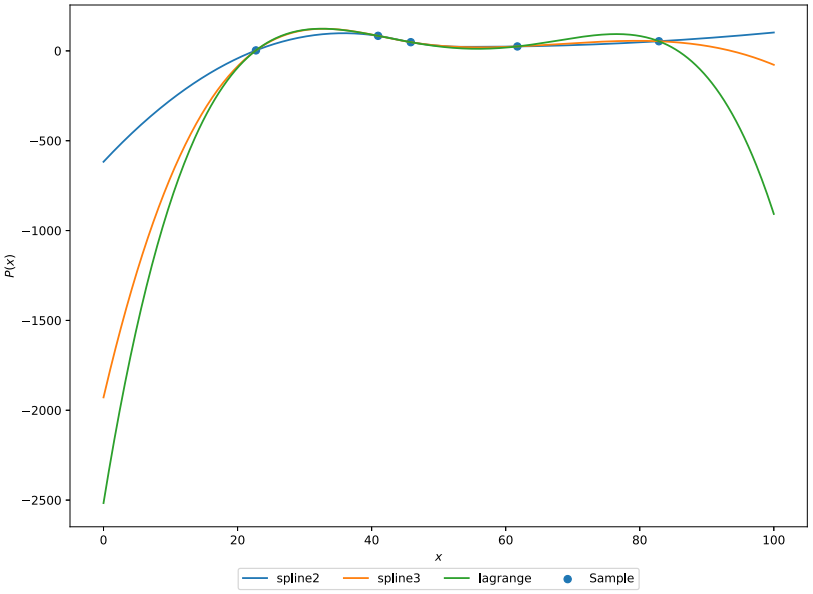
In [672]:
```
plot_polys([(p_lib_interp, "spline2"), (p_my_newton, "newton"), (p_my_lagrange, "lagrange")], x, y)
```



Funkcje sklejane

In [529]:
```
p_2spline = ip.interp1d(x, y, fill_value="extrapolate", kind="quadratic")
p_3spline = ip.interp1d(x, y, fill_value="extrapolate", kind="cubic")
```

In [673]:
```
plot_polys([(p_2spline, "spline2"), (p_3spline, "spline3"), (p_my_lagrange, "lagrange")], x, y)
```



Czasy obliczeń

In [531]:
```
def measure_my_lagrange(x, y):
    start = time.perf_counter_ns()
    my_lagrange(x, y)
    end = time.perf_counter_ns()
    return end - start

def measure_my_newton(x, y):
    start = time.perf_counter_ns()
    my_newton(x, y)
    end = time.perf_counter_ns()
    return end - start

def measure_quadratic(x, y):
    start = time.perf_counter_ns()
    ip.interp1d(x, y, fill_value="extrapolate", kind="quadratic")
    end = time.perf_counter_ns()
    return end - start

def measure_cubic(x, y):
    start = time.perf_counter_ns()
    ip.interp1d(x, y, fill_value="extrapolate", kind="cubic")
    end = time.perf_counter_ns()
    return end - start

def measure_lagrange(x, y):
    start = time.perf_counter_ns()
    ip.lagrange(x, y)
    end = time.perf_counter_ns()
    return end - start
```

In [648]:
```
df = pd.DataFrame(columns=['type', 'n', 'time'])

test_count = 10
points = range(4, 31)
for i in range(test_count):
    for n in points:
        xx = np.random.rand(n) * MAX
        yy = np.random.rand(n) * MAX
        df.loc[len(df)] = ['my_lagrange', n, float(measure_my_lagrange(xx, yy))]
        df.loc[len(df)] = ['my_newton', n, float(measure_my_newton(xx, yy))]
        df.loc[len(df)] = ['spline2', n, float(measure_quadratic(xx, yy))]
        df.loc[len(df)] = ['spline3', n, float(measure_cubic(xx, yy))]
        df.loc[len(df)] = ['lagrange', n, float(measure_lagrange(xx, yy))]

display(df.head(), df.tail())
```

	type	n	time
0	my_lagrange	4	2666500.0
1	my_newton	4	621700.0
2	spline2	4	442500.0
3	spline3	4	597300.0
4	lagrange	4	699500.0

	type	n	time
1345	my_lagrange	30	45464100.0
1346	my_newton	30	26262300.0
1347	spline2	30	295200.0
1348	spline3	30	303700.0
1349	lagrange	30	35497700.0

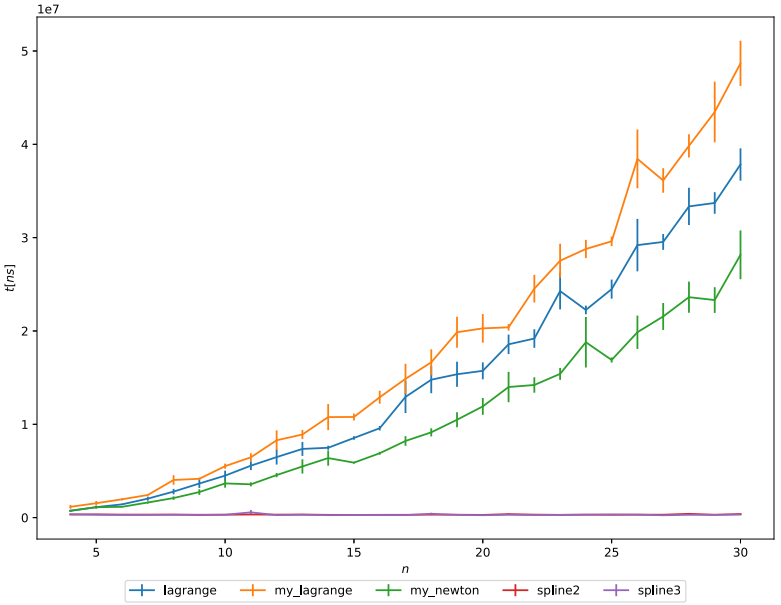
In [649]:
```
aggregated = df.groupby(['type', 'n'], as_index=False).agg(['mean', 'sem']).reset_index()
aggregated['mean'] = aggregated.time['mean']
aggregated['serr'] = aggregated.time['sem']
aggregated.drop('time', level=0, axis=1, inplace=True)
aggregated
```

Out[649]:

	type	n	mean	serr
0	lagrange	4	738470.0	9.305372e+04
1	lagrange	5	1125660.0	1.377599e+05
2	lagrange	6	1421080.0	7.508160e+04
3	lagrange	7	2034860.0	2.146058e+05
4	lagrange	8	2788850.0	2.892664e+05
5	lagrange	9	3654320.0	4.693349e+05
6	lagrange	10	4500090.0	5.349104e+05
7	lagrange	11	5566500.0	4.696283e+05
8	lagrange	12	6486290.0	8.006597e+05
9	lagrange	13	7361130.0	7.468532e+05
10	lagrange	14	7486310.0	2.214806e+05
11	lagrange	15	8535590.0	1.916560e+05
12	lagrange	16	9577970.0	2.393944e+05
13	lagrange	17	12932190.0	1.726611e+06
14	lagrange	18	14777800.0	1.447887e+06
15	lagrange	19	15360990.0	1.348207e+06
16	lagrange	20	15728540.0	9.060842e+05
17	lagrange	21	18569420.0	1.044582e+06
18	lagrange	22	19190020.0	9.947843e+05
19	lagrange	23	24275330.0	1.963576e+06
20	lagrange	24	22254430.0	4.635985e+05
21	lagrange	25	24482760.0	1.023824e+06
22	lagrange	26	29198620.0	2.808571e+06
23	lagrange	27	29540820.0	8.501232e+05
24	lagrange	28	33346630.0	1.998021e+06
25	lagrange	29	33716790.0	1.157054e+06
26	lagrange	30	37840620.0	1.730196e+06
27	my_lagrange	4	1158960.0	2.051884e+05
28	my_lagrange	5	1543430.0	2.406124e+05
29	my_lagrange	6	1963460.0	1.119836e+05
...	...	...	...	...
105	spline2	28	406260.0	6.712702e+04
106	spline2	29	307240.0	9.538066e+03
107	spline2	30	387600.0	3.729534e+04
108	spline3	4	368440.0	4.367120e+04
109	spline3	5	306680.0	3.392746e+04
110	spline3	6	286910.0	2.551898e+04
111	spline3	7	291320.0	2.387629e+04
112	spline3	8	299450.0	3.183398e+04
113	spline3	9	313750.0	3.573910e+04
114	spline3	10	307710.0	3.033995e+04
115	spline3	11	567070.0	2.580873e+05
116	spline3	12	268710.0	1.274380e+04
117	spline3	13	308460.0	3.424639e+04
118	spline3	14	264280.0	1.551991e+04
119	spline3	15	275690.0	2.461740e+04
120	spline3	16	283770.0	1.503353e+04
121	spline3	17	271180.0	2.227505e+04
122	spline3	18	412950.0	1.056309e+05
123	spline3	19	295170.0	3.541711e+04
124	spline3	20	260310.0	1.030425e+04
125	spline3	21	305040.0	2.990184e+04
126	spline3	22	281440.0	1.093224e+04
127	spline3	23	292720.0	2.501454e+04
128	spline3	24	310350.0	3.955668e+04
129	spline3	25	339690.0	3.801164e+04
130	spline3	26	322220.0	6.200235e+04
131	spline3	27	254280.0	6.881954e+03
132	spline3	28	300780.0	3.625540e+04
133	spline3	29	295520.0	2.704109e+04
134	spline3	30	323680.0	2.829328e+04

135 rows × 4 columns

In [674]:
```
figure(num=None, figsize=(11, 8), dpi=80, facecolor='w', edgecolor='k')
for typee in aggregated.type.unique():
    vdf = aggregated[aggregated.type == typee]
#     plt.plot(vdf['n'].values, vdf['mean'].values, label=typee)
    plt.errorbar(vdf['n'].values, vdf['mean'].values, yerr=vdf['serr'].values, label=typee)
plt.legend(bbox_to_anchor=(0., -.200, 1., .200), loc=10, ncol=5)
plt.xlabel('$n$')
plt.ylabel('$t[ns]$')
plt.show()
```



Efekt Rungego

In [18]:
```
SIZEb = 10
xb = np.array([i * (MAX / (SIZEb + 1)) for i in range(1, SIZEb + 1)])
yb = np.random.rand(SIZEb) * MAX
display(xb, yb)

p_2spline2 = ip.interp1d(xb, yb, fill_value="extrapolate", kind="quadratic")
p_3spline2 = ip.interp1d(xb, yb, fill_value="extrapolate", kind="cubic")
p_my_newton2 = my_newton(xb, yb)
plot_polys([(p_2spline2, "spline2"), (p_3spline2, "spline3"), (p_my_newton2, "newton")], xb, yb)
plt.ylim(-100, 200)
```

```
array([ 9.09090909, 18.18181818, 27.27272727, 36.36363636, 45.45454545,
       54.54545455, 63.63636364, 72.72727273, 81.81818182, 90.90909091])

array([36.20785889, 49.90876599, 66.24533513, 90.77271113, 24.36432586,
       24.15456288,  6.137739  , 29.83077774, 51.34712218, 24.64196059])
```

Out[18]:
```
(-100, 200)
```