MOWNIT - Laboratorium 9
Interpolacja Lagrange’a

Mikołaj Wróblewski

1. Za zadanie miałem napisanie funkcji interpolującej daną funkcję metodą Lagrange'a w n równoodległych węzłach, na początku było to n = 4. Dodatkowo za zadanie miałem przedstawić wielomian interpolujący na wykresie, wraz z funkcją interpolowaną. Dopisałem pomocniczą funkcję wyznaczającą n równoodległych punktów:

```
def equadistant(x0, x1, n):
    return [(x0 + i*(x1-x0)/(n-1)) for i in range(n)]
```

Następnie funkcja wykonująca interpolację:

```
def lagrange(f, x, n, nodes):
    values = list(map(f, nodes))
    sum = 0
    for i in range(n):
        term = 1
        for j in range(n):
            if i != j:
                term = term * (x - nodes[j]) / (nodes[i] - nodes[j])
        term = term*values[i]
        sum += term
    return sum
```

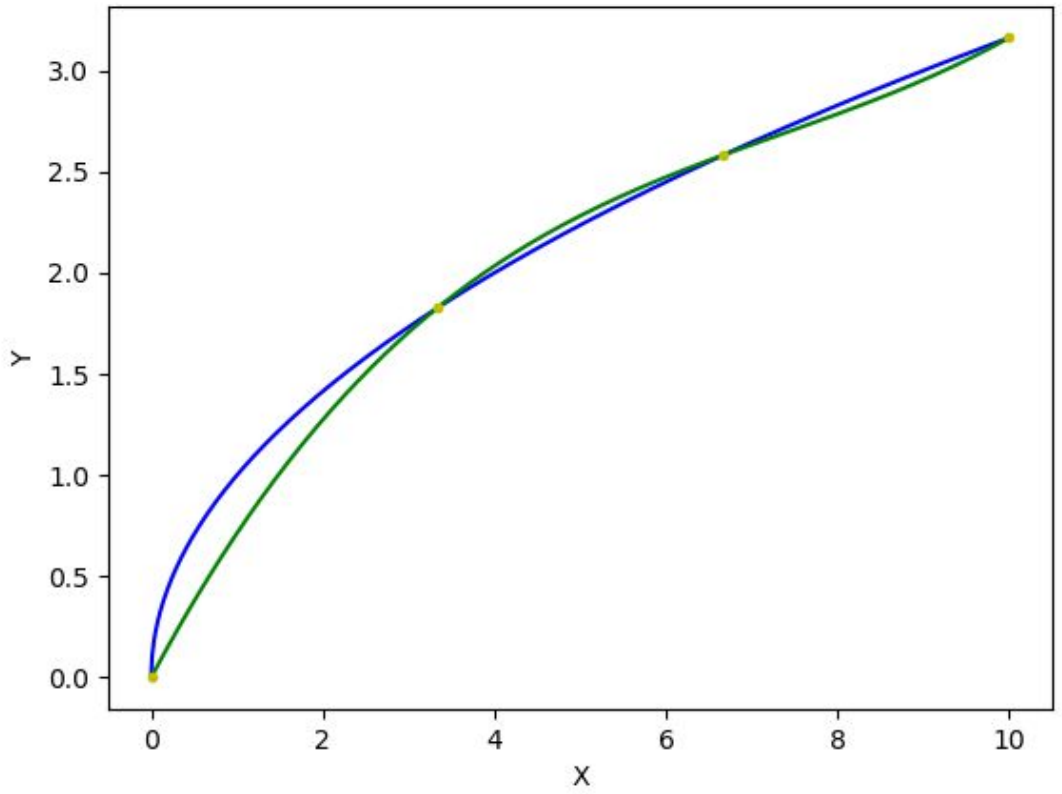
Gdzie f to funkcja interpolowana, x dla którego chcemy policzyć wartość z wielomianu, n - liczba węzłów, nodes - węzły.

Funkcję testującą można zobaczyć w pliku lagrange.py dołączoną do archiwum z rozwiązaniem, nie wrzucam jej kodu tutaj dla czytelności sprawozdania, powiem o niej tylko tyle, że na terminal zostają wypisane punkty znajdujące się w połowie odległości między kolejnymi węzłami interpolacji, wraz z wartością błędu interpolacji w tych punktach.

Tutaj wykresy dla 4 punktów interpolacji dla kolejnych funkcji, kolorem niebieskim oznaczona jest funkcja interpolowana, a kolorem zielonym funkcja interpolująca.

F = sqrt(x)

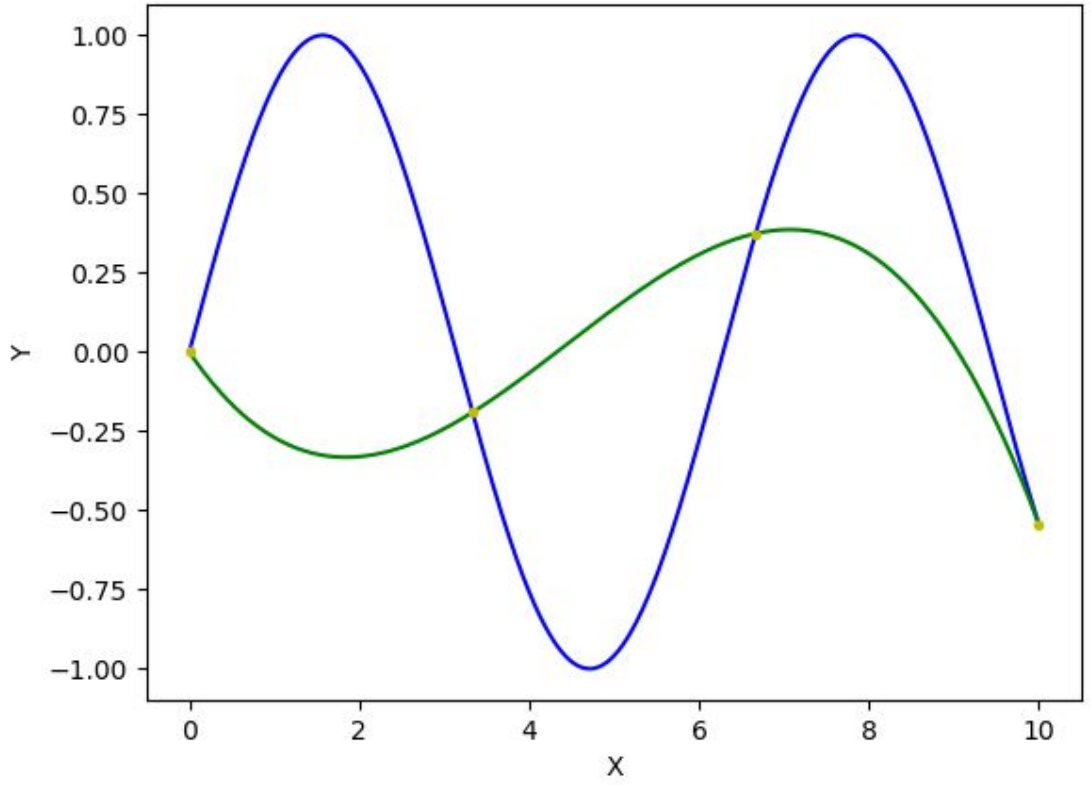
x : [1.6666666666666667, 5.0, 8.333333333333334]
error : [0.18859063323151615, 0.04563821888965469, 0.048469316500423076]



Największy błąd możemy zaobserwować w pierwszym ze sprawdzanych punktów, widać także na wykresie, że w tym miejscu funkcja jest najgorzej przybliżana, w dalszych przedziałach jest już znacznie lepiej.

F = sin(x)

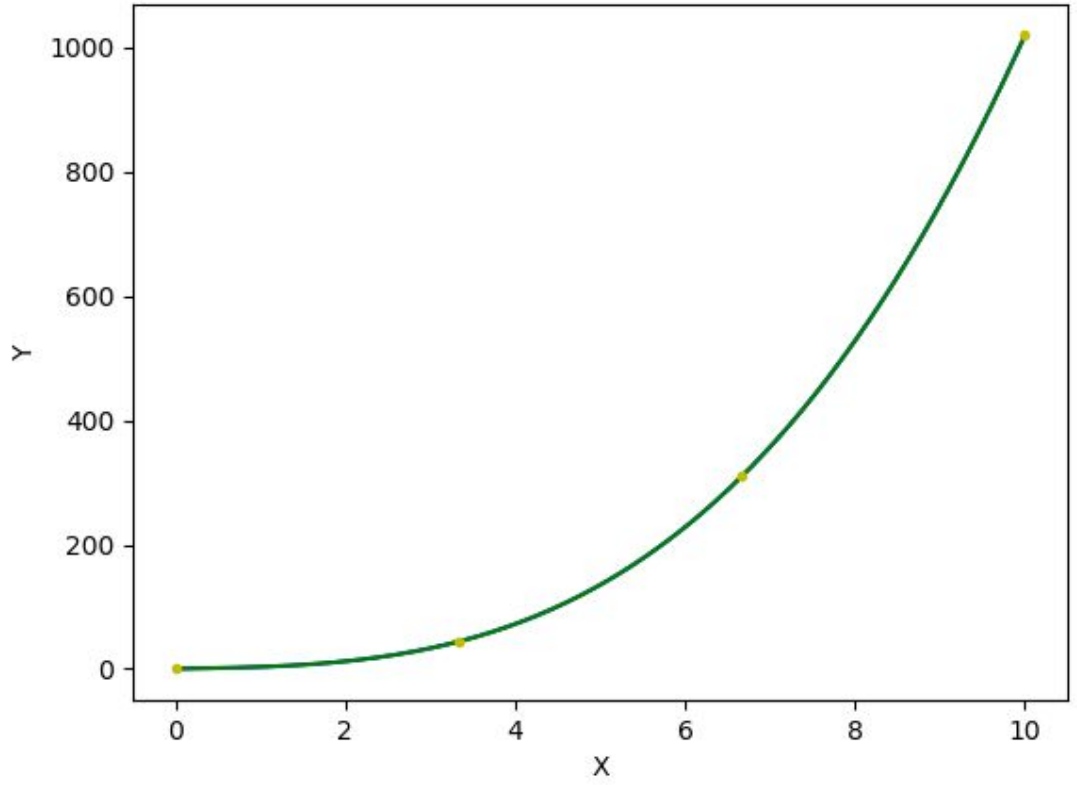
x : [1.6666666666666667, 5.0, 8.333333333333334]
error : [1.3249890019316244, 1.0961911821725747, 0.6469814381885147]



W tym przypadku mamy już znacznie gorsze przybliżenie, nasze błędy są także znacznie większe.

F = x^3 + 2x

x : [1.6666666666666667, 5.0, 8.333333333333334]
error : [5.329070518200751e-15, 0.0, 0.0]



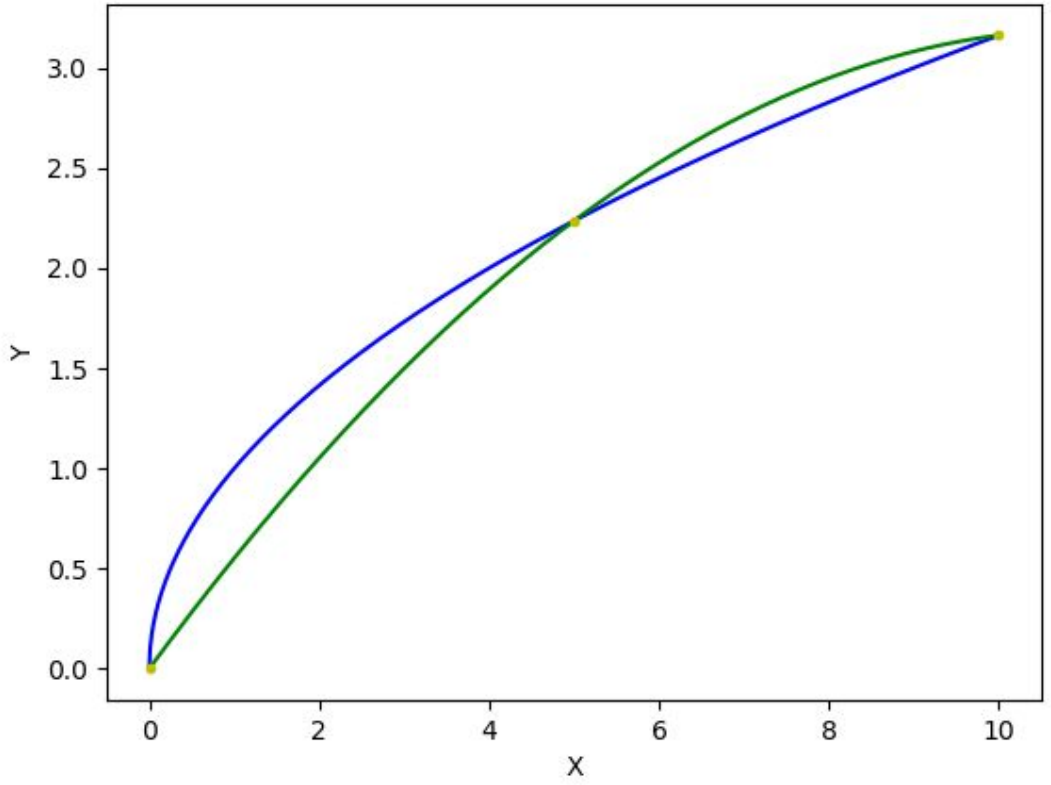
Jak widzimy funkcja została przybliżona praktycznie perfekcyjnie, błędy w badanych punktach są idealne. Oczywiście znając wzór na błąd interpolacji Lagrange'a nie jest to dla mnie żadnym zaskoczeniem, omówię go jednak na koniec tego sprawozdania.

Następnie interpolację powtórzyłem dla 3,5 i 8 węzłów interpolacji.

2. N = 3

F = sqrt(x)

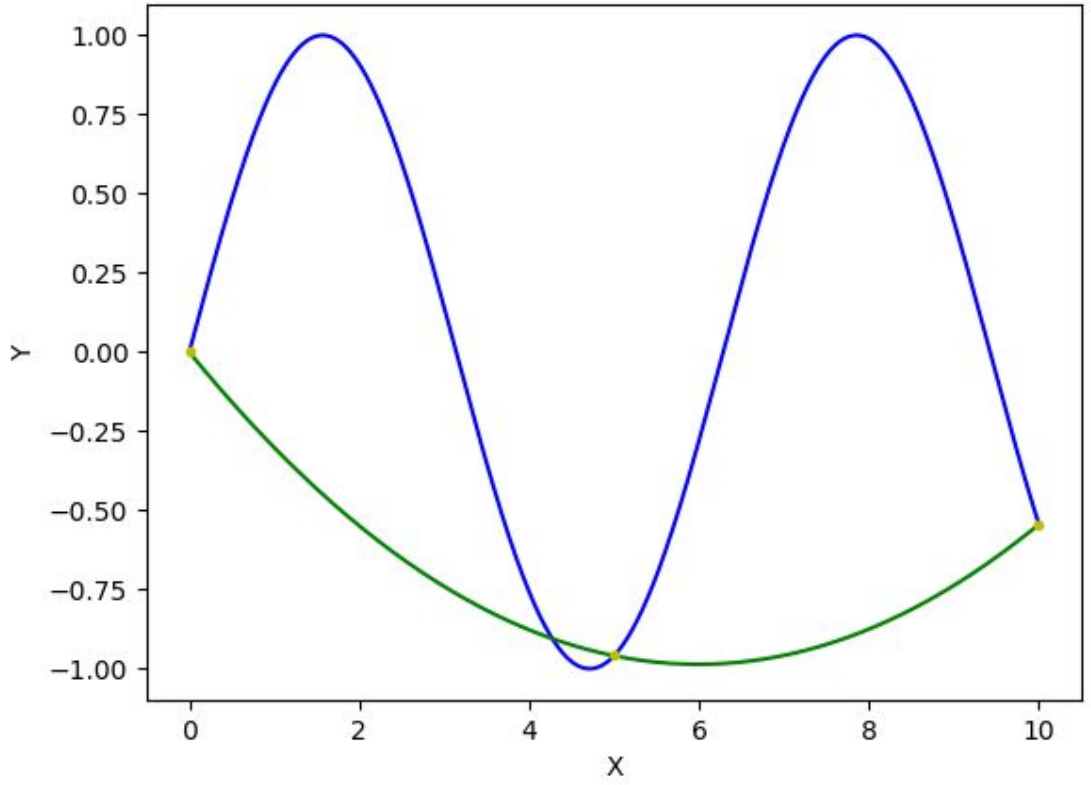
x : [2.5, 7.5]
error : [0.2993725544803949, 0.12429231816215403]



Widzimy znacznie słabsze przybliżenie naszej funkcji, błąd w badanych punktach również się zwiększył.

F = sin(x)

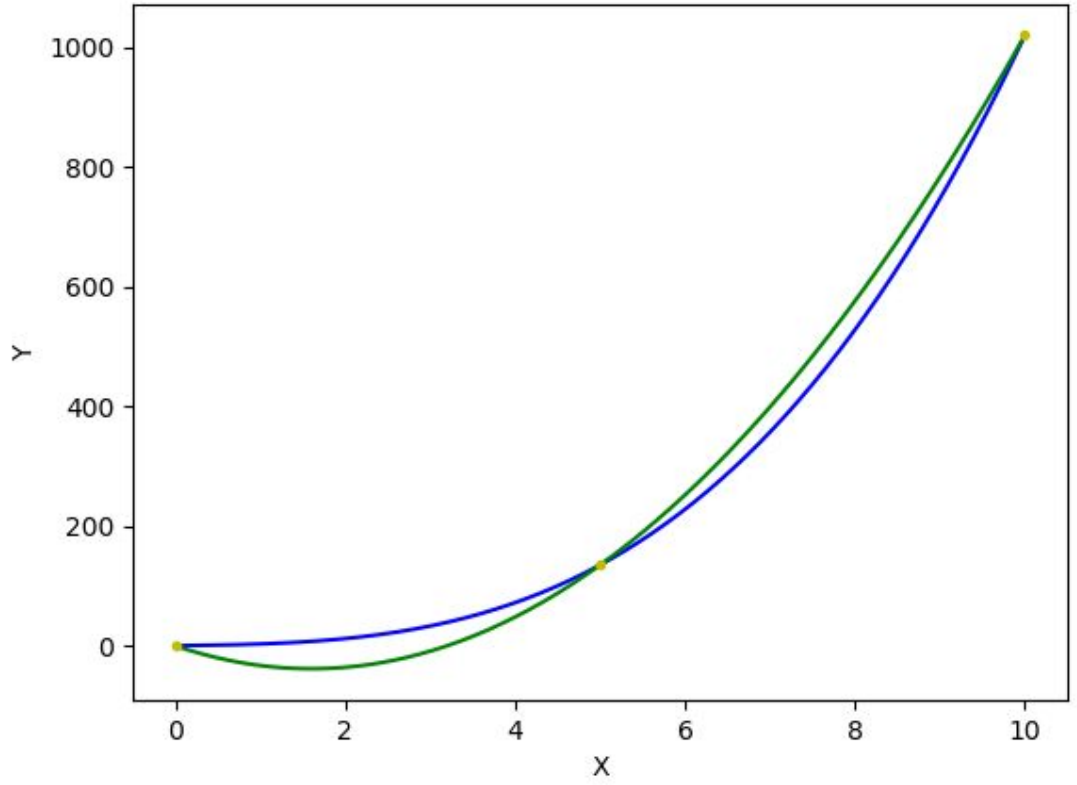
x : [2.5, 7.5]
error : [1.249662711240139, 1.8612010993556063]



Widzimy, że funkcja interpolująca nie jest prawie żadnym przybliżeniem funkcji interpolowanej. Błędy w badanych mają również duże wartości.

F = x^3 + 2x

x : [2.5, 7.5]
errors : [46.875, 46.875]

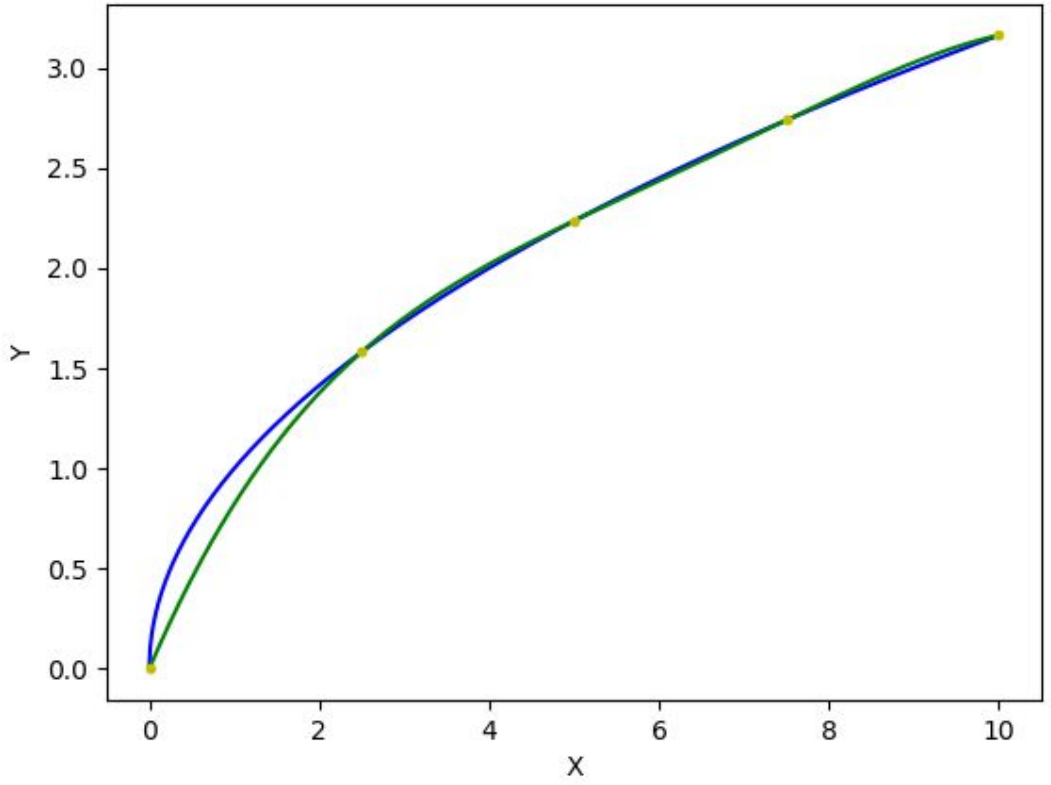


Widzimy, że pojawiły się dosyć duże błędy. Skąd, zostanie to przeze mnie omówione w dalszej części. Mimo to na wykresie interpolacja nie wygląda tak źle - oczywiście jest to kwestia skali.

3. N = 5

F = sqrt(x)

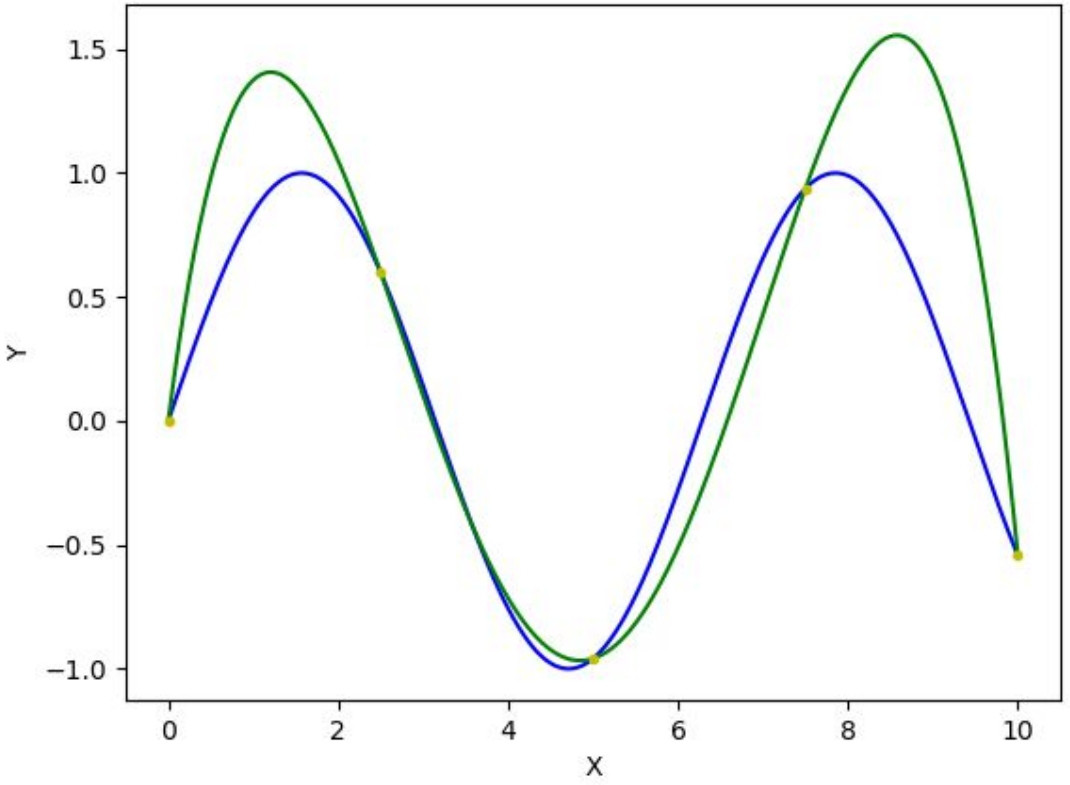
x : [1.25, 3.75, 6.25, 8.75]
errors : [0.1359679923695617, 0.023110084787080654, 0.014619372468709368, 0.025027586394580137]



Widzimy, że największy błąd nadal znajduje się w pierwszym przedziale. Jednakowoż dla dalszych przedziałów funkcja interpolująca świetnie przybliża funkcję interpolowaną.

F = sin(x)

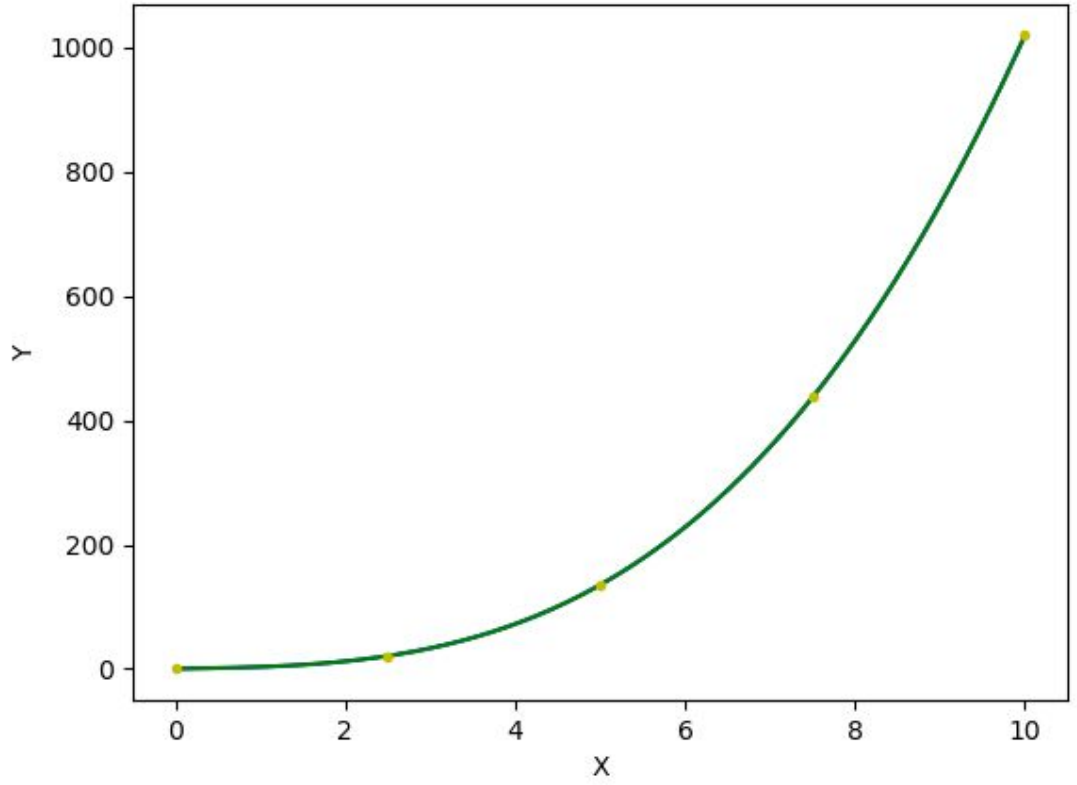
x : [1.25, 3.75, 6.25, 8.75]
error : [0.45644432052811035, 0.018538514511031612, 0.27363737283393086, 0.9077852425635107]



Jak widać po błędach w testowanych punktach oraz po samym wykresie, w porównaniu do wcześniejszych interpolacji tejże funkcji, funkcja interpolująca lepiej przybliża funkcję interpolowaną.

F = 3^x + 2x

x : [1.25, 3.75, 6.25, 8.75]
error : [0.0, 0.0, 0.0, 1.1368683772161603e-13]

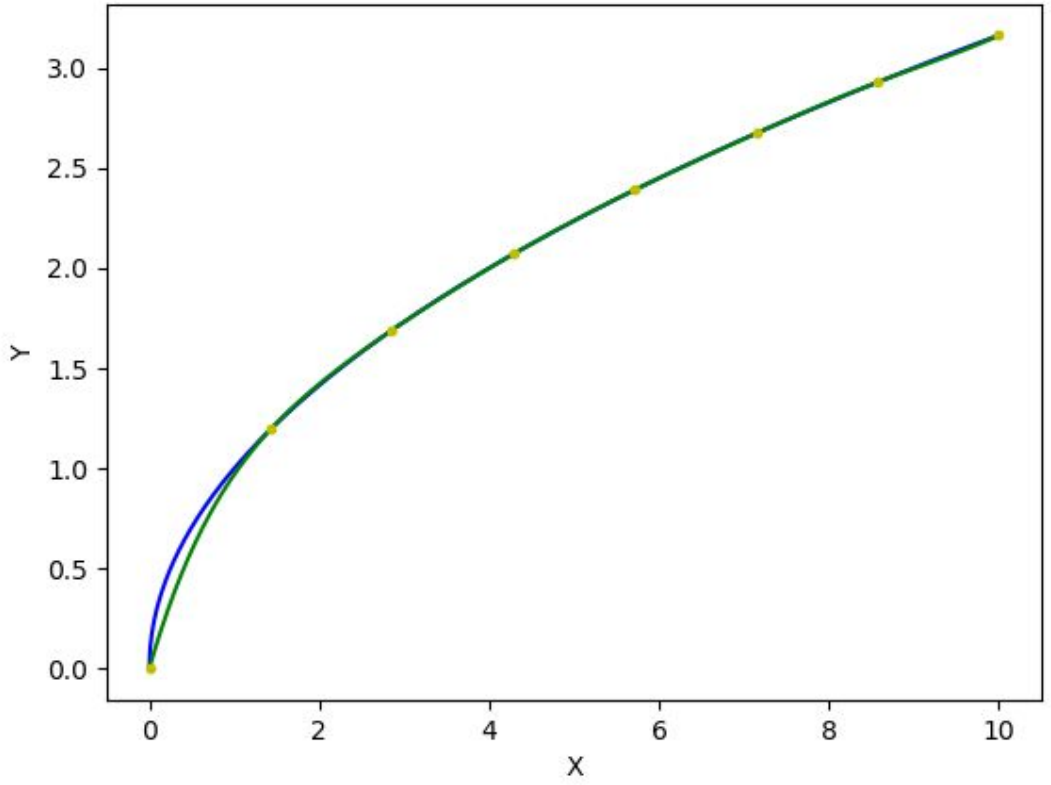


W tym przypadku funkcja interpolująca przybliża - można śmiało rzec - idealnie. Oczywiście nadal nie jest to żadne zaskoczenie. W przypadku 8 punktów będzie tylko lepiej.

4. N = 8

F = sqrt(x)

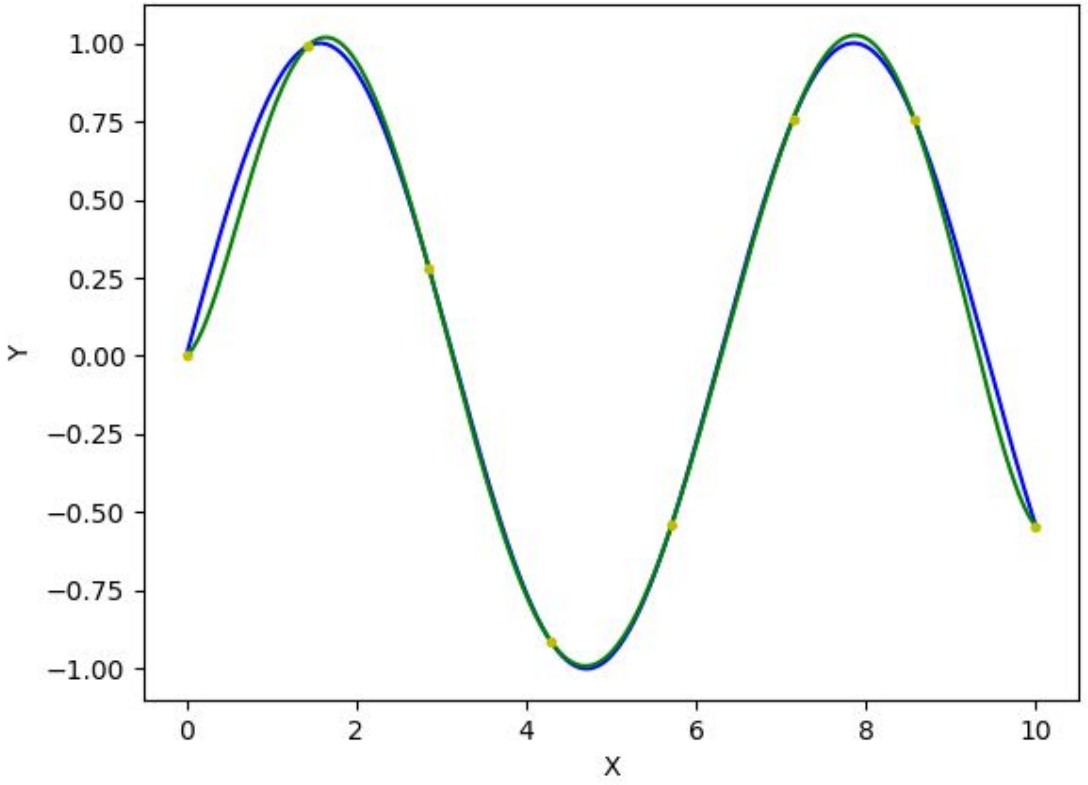
x : [0.7142857142857143, 2.142857142857143, 3.5714285714285716, 5.0, 6.428571428571429, 7.857142857142857, 9.285714285714286]
errors : [0.0722426411608843, 0.006443365321250338, 0.0018336486322845236, 0.0010401866750062716, 0.0010532933156270907, 0.0019118797854735803, 0.007052499299445625]



W tym przypadku funkcja jest przybliżana bardzo dobrze. Dla jeszcze większej ilości węzłów byłoby oczywiście jeszcze lepiej.

F = sin(x)

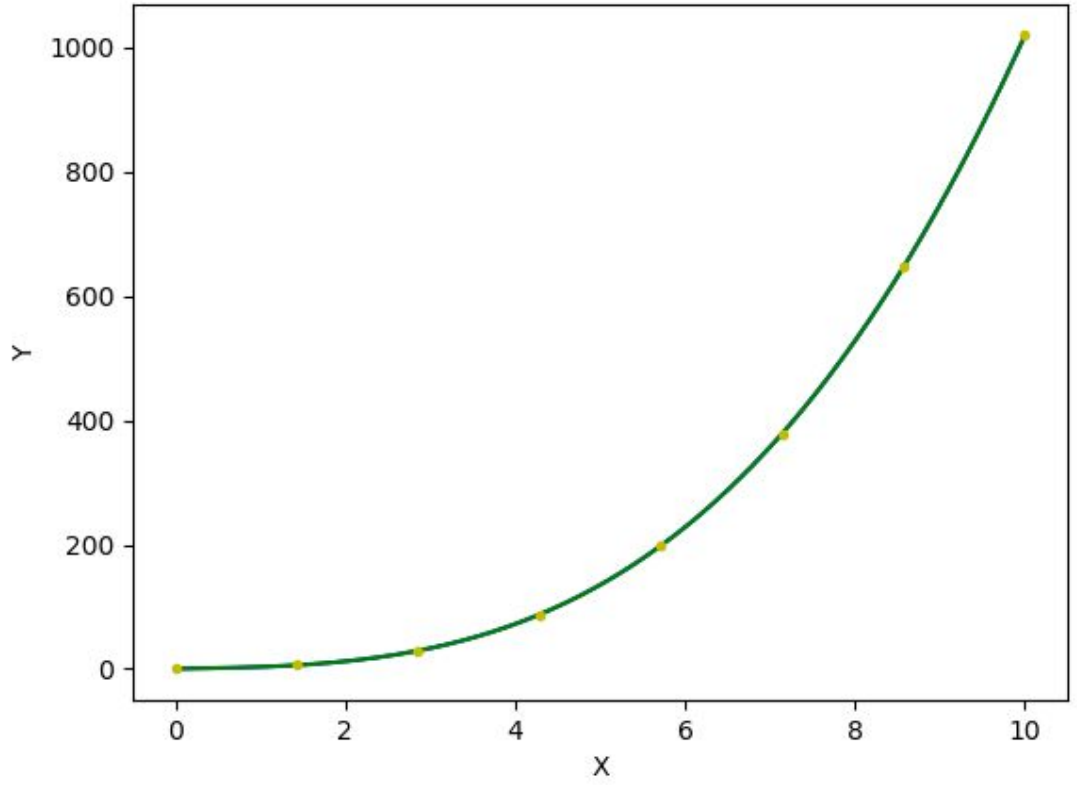
x : [0.7142857142857143, 2.142857142857143, 3.5714285714285716, 5.0, 6.428571428571429, 7.857142857142857, 9.285714285714286]
error : [0.12459386788914195, 0.031018723297912, 0.014507078297654352, 0.011016098996751267, 0.01308191469789241, 0.025121332247741712, 0.08988241404141506]



Jak widzimy funkcja sinus w tym przypadku jest jeszcze lepiej przybliżana. Głębsze omówienie czemu się tak dzieje - na końcu sprawozdania.

F = x^3 + 2x

x : [0.7142857142857143, 2.142857142857143, 3.5714285714285716, 5.0, 6.428571428571429, 7.857142857142857, 9.285714285714286]
error : [1.9539925233402755e-14, 3.552713678800501e-15, 1.4210854715202004e-14, 2.842170943040401e-14, 5.684341886080802e-14, 1.7053025658242404e-13, 1.1368683772161603e-13]



Nasza funkcja w badanych punktach ma błędy przybliżenia rzędów 1e-13 w najgorszym przypadku, jak widzimy interpolacja Lagrange'a świetnie radzi sobie z wielomianem.

5. Błąd interpolacji Lagrange'a

Tak jak napisałem wcześniej na końcu tego sprawozdania zajmę się błędem interpolacji. Czemu funkcja wielomianowa jest ze wszystkich tych funkcji przybliżana najlepiej? Otóż odpowiedzią na to jest tenże wzór:

$$E_n(x) = \frac{f^{n+1}(\xi(x))}{(n+1)!} \times (x - x_0)(x - x_1)\dots(x - x_n)$$

Wzór na błąd interpolacji Lagrange'a jasno nam mówi, czemu funkcja ostatnia jest przybliżana najlepiej. Wiąże się z to z n+1 pochodną tej funkcji, gdzie n stopniem wielomianu interpolującego. Liczba n+1 jest też liczbą węzłów interpolacji.

Co do oszacowania błędu, jesteśmy w stanie wyliczyć jego górną granicę, ponieważ nic nie wiemy o funkcji

$$\xi(x)$$

Jak zatem to wykonać?

Możemy znaleźć błędy w poszczególnych punktach na przedziale, na którym interpolujemy, postępując następująco:

Wyliczamy maksymalną wartość pochodnej n+1 stopnia na danym przedziale. Następnie wyliczamy maksymalną wartość bezwględną z tego wyrażenia:

$$(x - x_0)(x - x_1)\dots(x - x_n)$$

na przedziale, na którym interpolujemy. Możemy na przykład spróbkować 1000 punktów. Następnie podstawiamy do wzoru kolejne części, otrzymujemy górną granicę błędu naszej interpolacji. Niestety napisanie funkcji obliczającej górną wartość błędu w kodzie nie udało mi się, z uwagi na to w jaki sposób dostępne pakiety liczą pochodną. Natomiast patrząc na wzór możemy od razu oszacować wielkość błędu trzeciej funkcji, skoro przy czwartej pochodnej funkcja f = x ^ 3 + 2x się zeruje, to błąd będzie niesamowicie mały (teoretycznie powinien być równy zero, ale dochodzi również reprezentacja zmiennoprzecinkowa i błędy z nią związane). Natomiast widzimy, że w przypadku funkcji sinus, zawsze dostaniemy funkcję będącą sinusoidą lub cosinusoidą (minus nie jest tutaj istotny). Wtedy to iloczyn stojący powyższy wraz z silnią w mianowniku decydują o granicy górnej błędu.