Metody Obliczeniowe w Nauce i Technice

Laboratorium 5 – raport

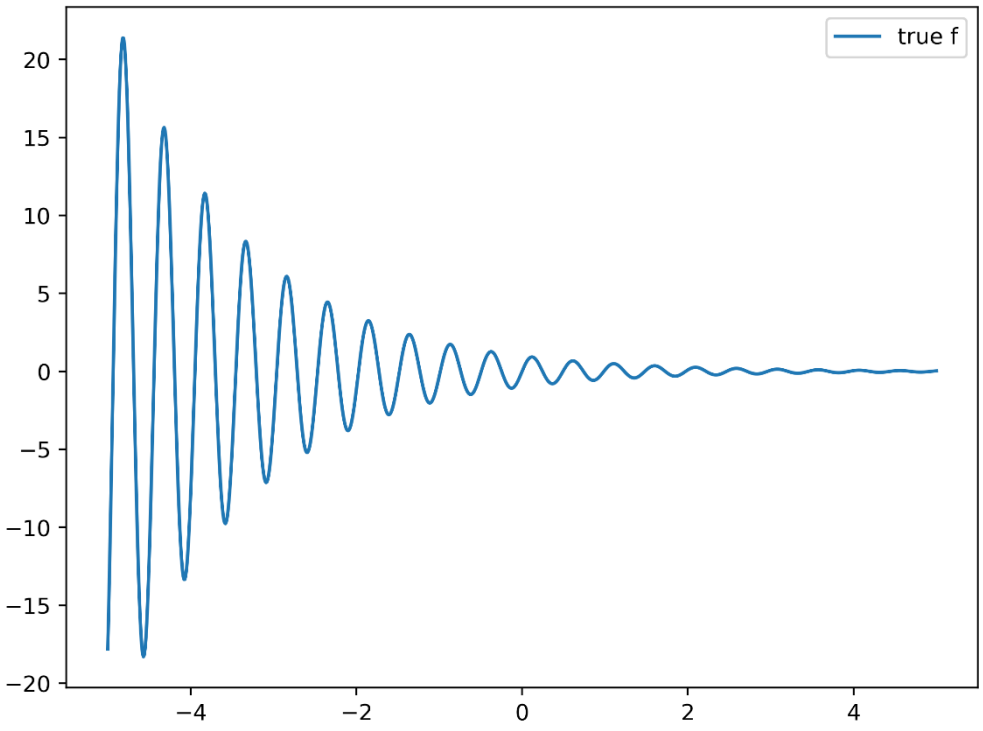
Mateusz Kocot

1. Funkcje testowa

Do testowania aproksymacji wykorzystano dwie funkcje przedstawione poniżej. Funkcje testowano na przedziale $[-5,\ 5]$. W celu zobrazowania przebiegów obliczono wartości funkcji dla 1001 punktów oddalonych od siebie o 0.01. Przebiegi zwizualizowano z wykorzystaniem języka Python oraz pakietu Matplotlib.

$$1.\ f(x) = \sin\left(\frac{40x}{\pi}\right) e^{\frac{-2x}{\pi}}$$

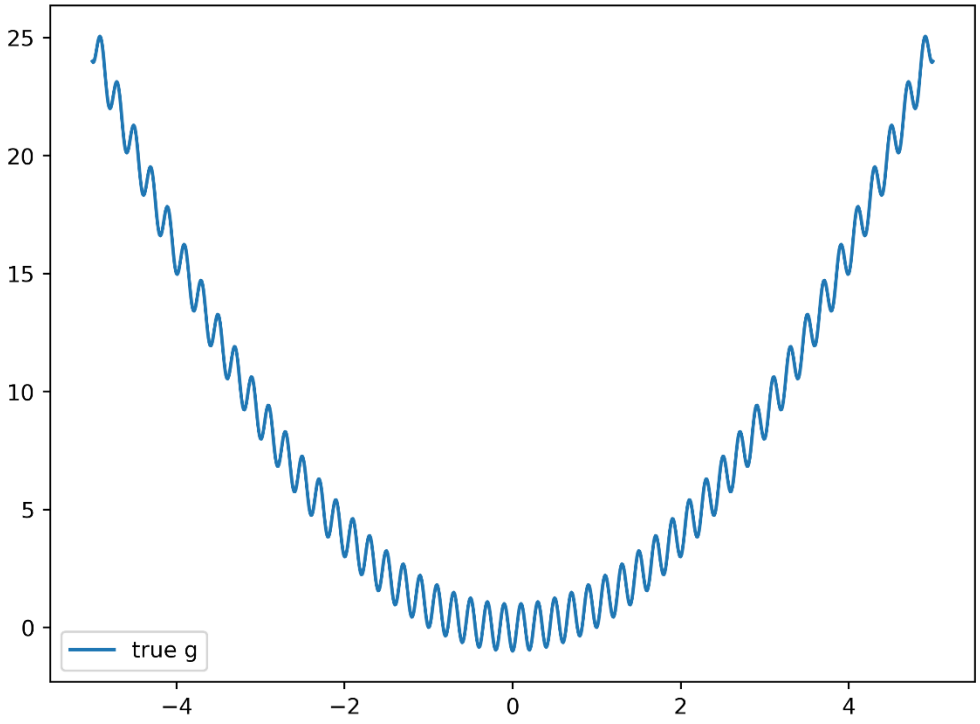
```
inline double f(double x)
{
    return sin(40 * x / M_PI) * exp(-2 * x / M_PI);
}
```



Przebieg funkcji f na przedziale $[-5, 5]$.

$$2.\ g(x) = x^2 - \cos(10\pi x)$$

```
inline double g(double x)
{
    return x * x - cos(10 * M_PI * x);
}
```



Przebieg funkcji g na przedziale $[-5, 5]$.

2. Aproksymacja średniokwadratowa wielomianami algebraicznymi

Wielomian interpolujący wyznaczany był w taki sposób, by przechodził przez wszystkie punkty z danego zbioru. Zadanie funkcji aproksymującej jest jednak inne. Dla każdej funkcji z danego zbioru szukamy jej współczynnika tak, żeby suma tych wszystkich funkcji pomnożonych przez odpowiednie współczynniki w jak najlepszy sposób obrazowała relację między współrzędnymi x i y punktów wejściowych. Funkcja ta nie musi jednak przechodzić przez wszystkie punkty. Zadaniem aproksymacji jest minimalizacja błędu.

Metoda aproksymacji średniokwadratowej wielomianami algebraicznymi polega na minimalizacji sumy kwadratów różnic pomiędzy prawdziwymi wartościami pewnej funkcji a wartościami wielomianu aproksymującego. Musimy zatem, Dla n danych punktów $x_0, \dots, x_{n-1}$, ich wartości $F(x_0), \dots, F(x_{n-1})$ oraz stopnia wielomianu m, znaleźć minimum funkcji:

$$H(a_0, \dots, a_m) = \sum_{i=0}^{n-1} \left[F(x_i) - \sum_{j=0}^{m} a_j {x_i}^j \right]^2$$

Szukamy współczynników $a_0, \dots, a_m$ wielomianu. W tym celu dla każdego a_k $(k = 0, \dots, m)$ musimy obliczyć pochodną cząstkową:

$$\frac{\partial H}{\partial a_k} = -2 \sum_{i=0}^{n-1} \left[F(x_i) - \sum_{j=0}^{m} a_j {x_i}^j \right] {x_i}^k$$

W celu znalezienia minimum, pochodną należy porównać z 0:

$$\sum_{i=0}^{n-1}\left[F(x_i)-\sum_{j=0}^{m}a_j{x_i}^j\right]{x_i}^k=0$$

Po wymnożeniu i odpowiednich przekształceniach otrzymujemy:

$$\sum_{j=0}^{m}\left(\sum_{i=0}^{n-1}{x_i}^{j+k}\right)a_j=\sum_{i=0}^{n-1}F(x_i){x_i}^k \tag{1}$$

Niech

$$g_{k,j}=\sum_{i=0}^{n-1}{x_i}^{j+k}$$

$$b_k=\sum_{i=0}^{n-1}F(x_i){x_i}^k$$

Wówczas równanie (1) możemy zapisać w postaci macierzowej:

$$\mathrm{G}\cdot\mathrm{A}=\mathrm{B} \tag{2}$$

Szukamy zatem macierzy współczynników A.

Takie równanie macierzowe można obliczyć np. metodą eliminacji Gaussa. W tym celu wykorzystano nieco zmodyfikowaną funkcję *gaussSolve* zaimplementowaną przy okazji laboratorium 2. Dalsze funkcje będą także wykorzystywać prostą klasę *Point* z ostatniego laboratorium, której zadaniem jest reprezentacja dwuwymiarowego punktu.

Zaimplementowano funkcję *algebraicLeastSquares* znajdującą współczynniki wielomianu aproksymującego średniokwadratowo.

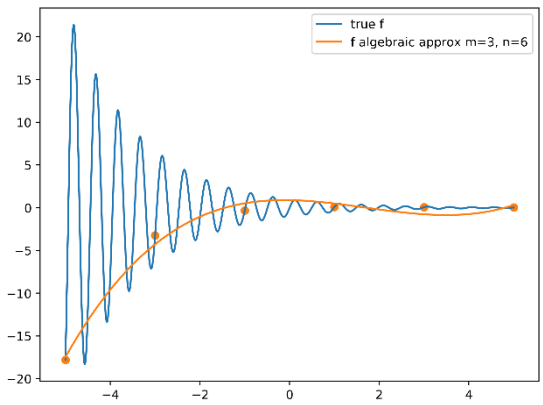
```
std::vector<double> algebraicLeastSquares(std::vector<Point> points, int m)
{
    m++;
    int n = points.size();

    std::vector< std::vector<double> > G(m, std::vector<double>(m));
    for (int k = 0; k < m; k++)
    {
        for (int j = 0; j < m; j++)
        {
            for (int i = 0; i < n; i++)
            {
                G[k][j] += std::pow(points[i].getX(), j + k);
            }
        }
    }

    std::vector<double> B(m);
```

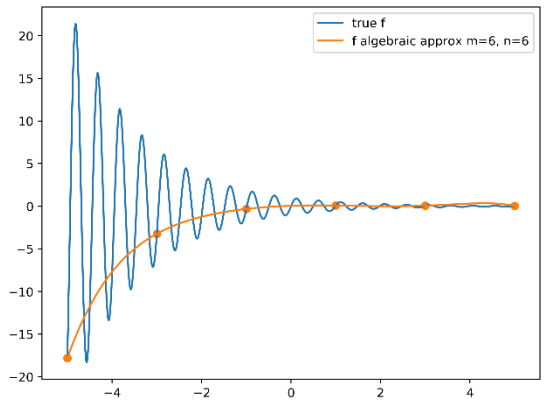
```
    for (int k = 0; k < m; k++)
    {
        for (int i = 0; i < n; i++)
        {
            B[k] += points[i].getY() * std::pow(points[i].getX(), k);
        }
    }

    return(gaussSolve(G, B));
}
```

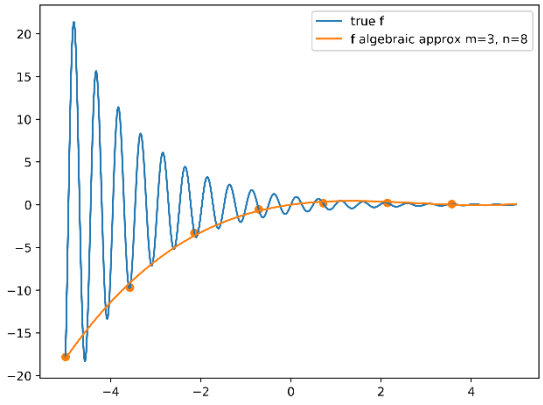
Funkcja ta przyjmuje tablicę punktów *points* oraz stopień m wielomianu aproksymującego. Funkcja zwraca tablicę $m + 1$ współczynników wielomianu.

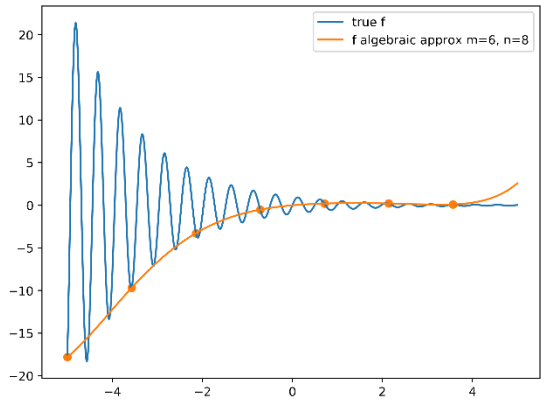
Funkcja najpierw według wcześniej napisanych wzorów oblicza macierze G i B, a następnie wykorzystuje metodę eliminacji Gaussa, żeby znaleźć tablicę współczynników A.

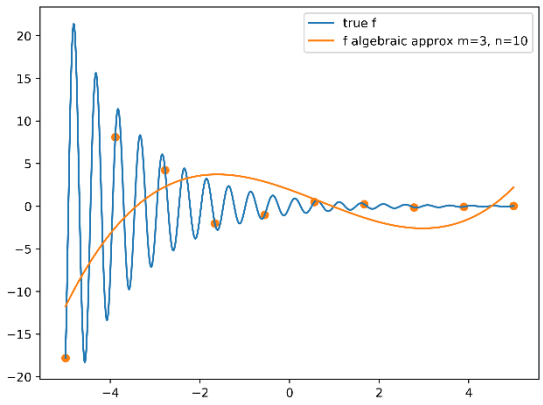
Funkcje testowe będę przybliżał wielomianami 3 oraz 6 stopnia dla różnych liczb równoodległych punktów. Najpierw przedstawię wyniki eksperymentów dla pierwszej funkcji testowej f.

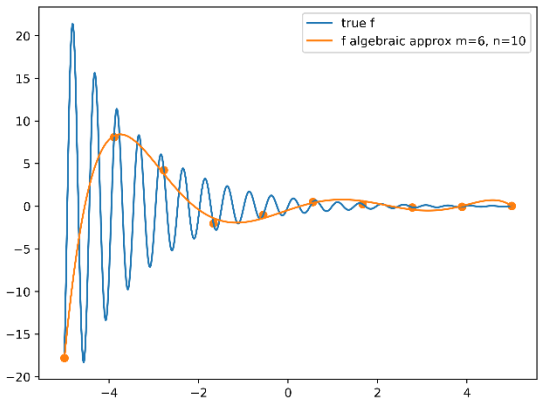


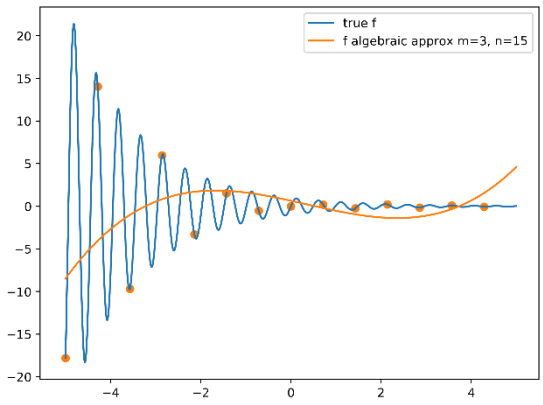


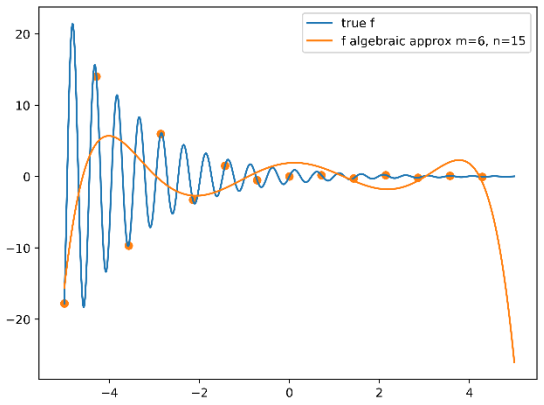


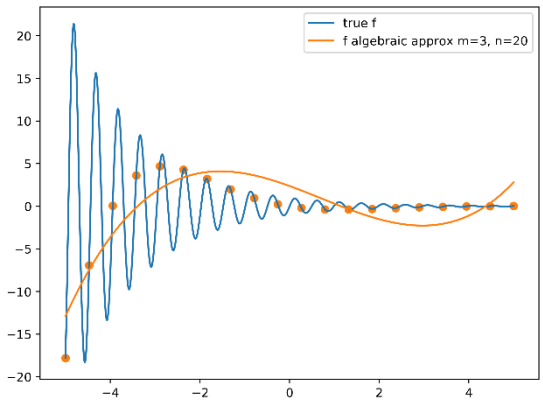


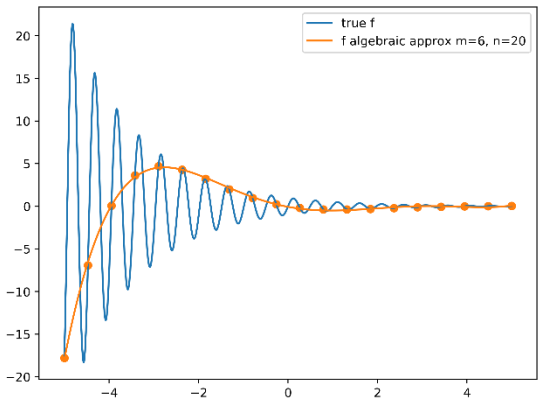

true f
f algebraic approx m=3, n=10

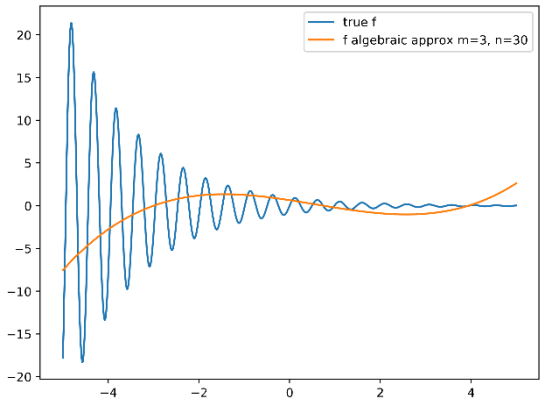

true f
f algebraic approx m=6, n=10

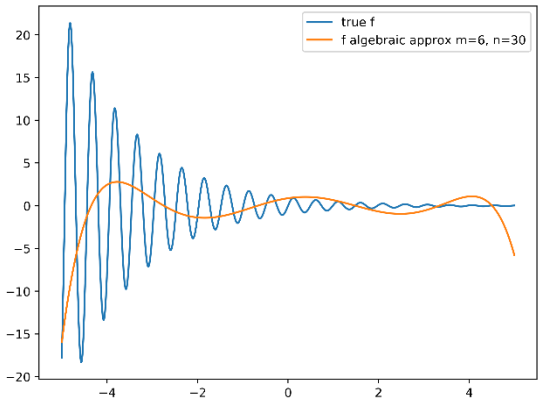

true f
f algebraic approx m=3, n=15

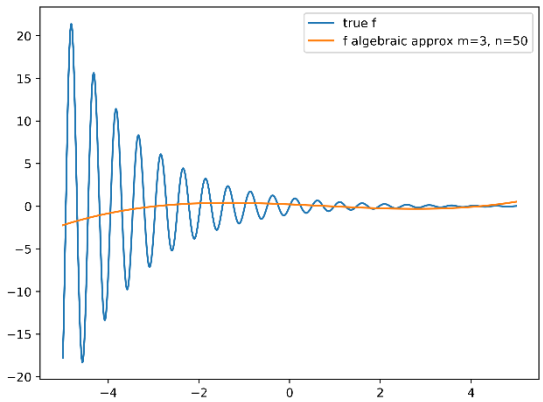

true f
f algebraic approx m=6, n=15

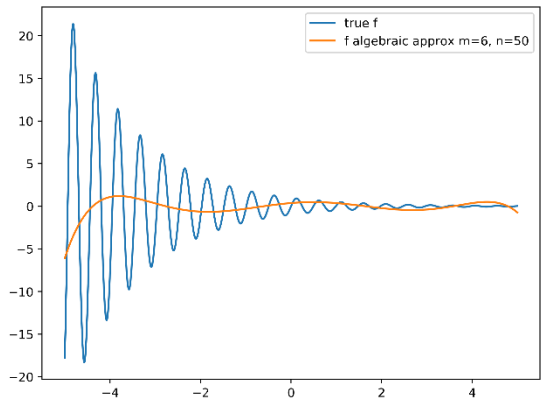

true f
f algebraic approx m=3, n=20

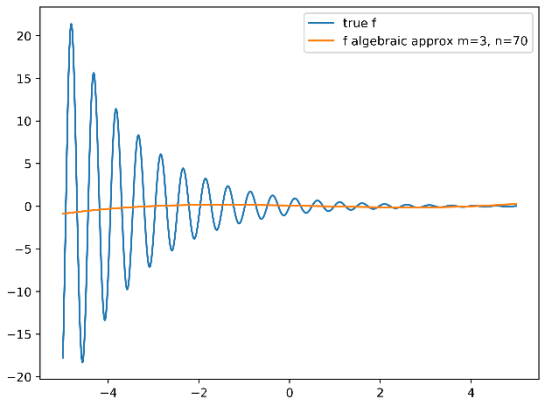

true f
f algebraic approx m=6, n=20

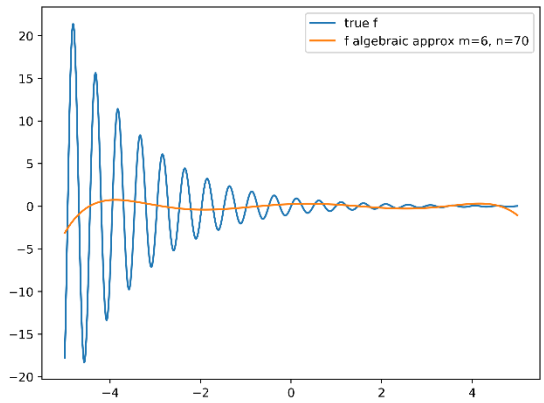

true f
f algebraic approx m=3, n=30

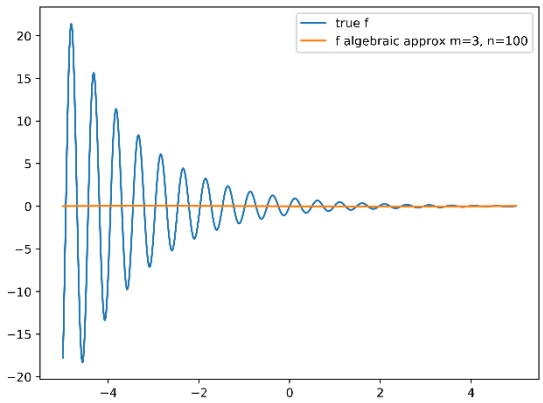

true f
f algebraic approx m=6, n=30

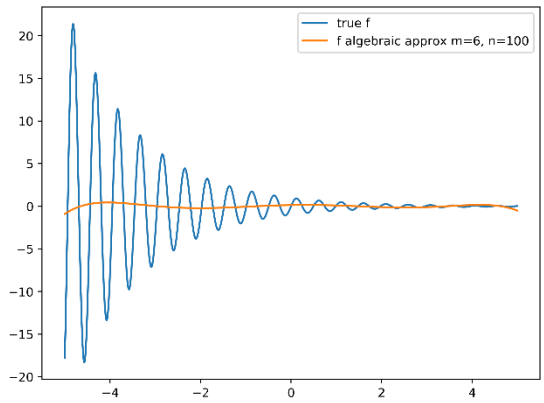












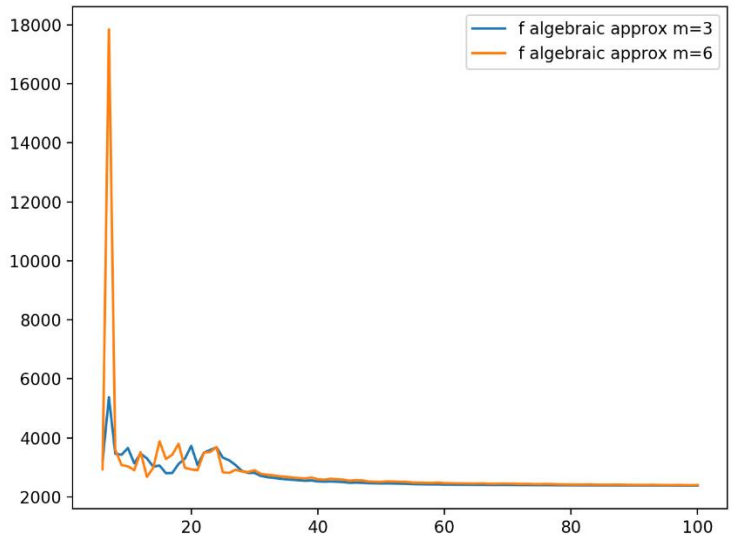
Aproksymacje wielomianami 3 i 6 stopnia są podobne, jednak wielomian 6 stopnia jest szybciej zmienny. Jest to spowodowane faktem, iż im większego stopnia jest wielomian, tym więcej razy może on zmieniać monotoniczność.

W wypadku tej funkcji stopień wielomianu zależy od tego co chcemy uzyskać. Jeżeli potrzebna jest nam jak najdokładniejsza aproksymacja, lepiej wybrać wielomian dużego stopnia. Wówczas jednak jeszcze lepiej sprawdziłaby się interpolacja.

Aproksymacja jednak jest najczęściej wykorzystywana, gdy mamy dane obarczone jakimś błędem. W przypadku funkcji f mogą to być np. dane mierzone z coraz mniejszym błędem. Najczęściej mamy też pojęcie, jak może wyglądać funkcja, potrzebne są tylko te współczynniki. Jeżeli dysponowalibyśmy wiedzą o tym, że funkcja f powinna być prostą, najlepiej byłoby wykorzystać aproksymację wielomianem 1 stopnia.

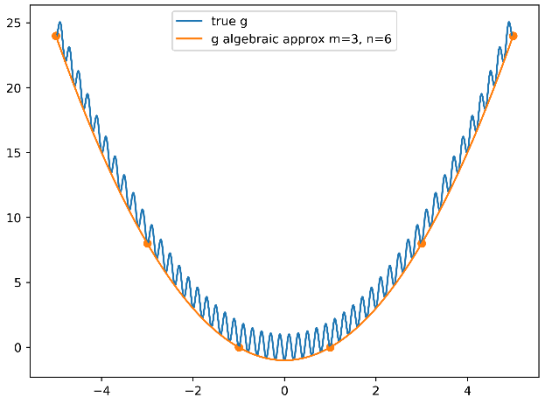
Jak widać na rysunku, dla 100 punktów, wielomian 3 stopnia dostosował się do punktów i jest to praktycznie prosta. Wielomian 6 stopnia nawet dla 100 punktów ma kilka punktów, w których zmienia monotoniczność. Efekt ten jest oczywiście powiększony przez fakt, iż funkcja f jest coraz bardziej „rozpięta", gdy maleje x.

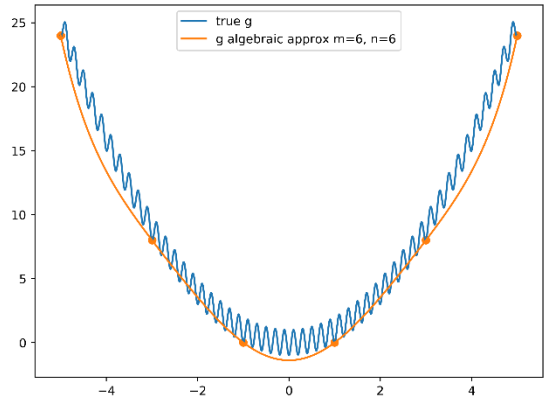
Wykonano także wykres błędów jako sumy różnic pomiędzy wartością prawdziwą a wartością wielomianu:

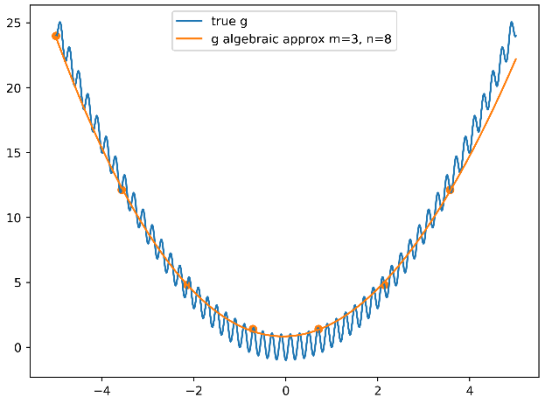


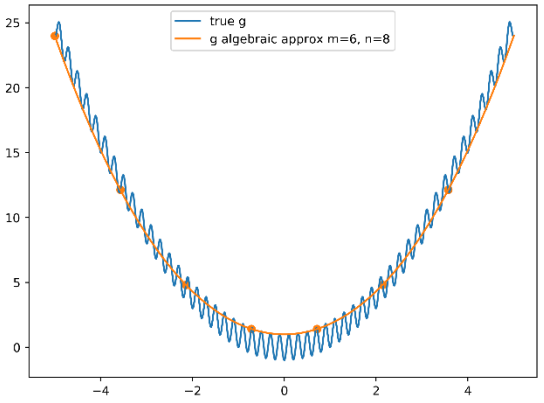
Na wykresie widać, że ze względu na gorsze dopasowanie, błąd wielomianu 6 stopnia jest na początku dużo większy od błędu wielomianu stopnia 3. Jednak, gdy punktów jest już wystarczająco wiele (30), wyniki błędów dla obu wielomianów mniej więcej się zrównują.

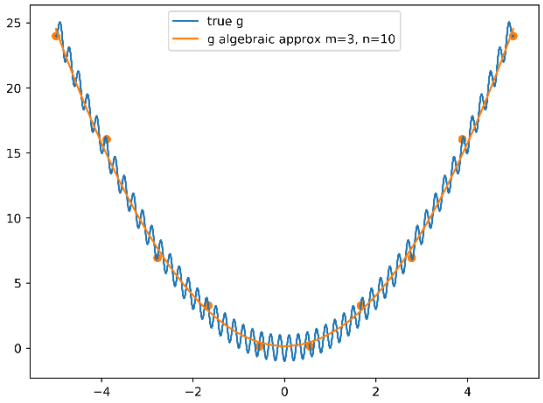
Przejdźmy teraz do funkcji g.

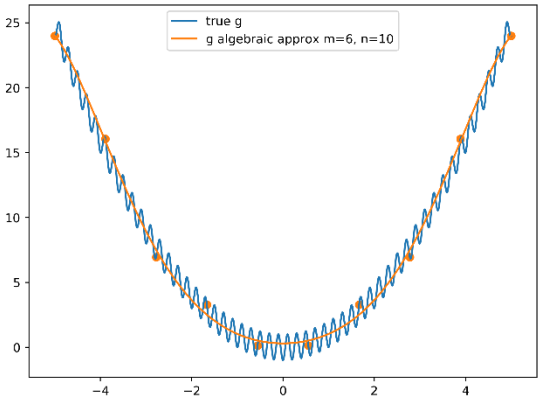


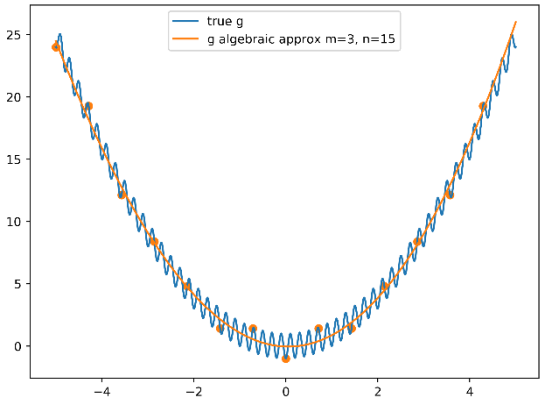


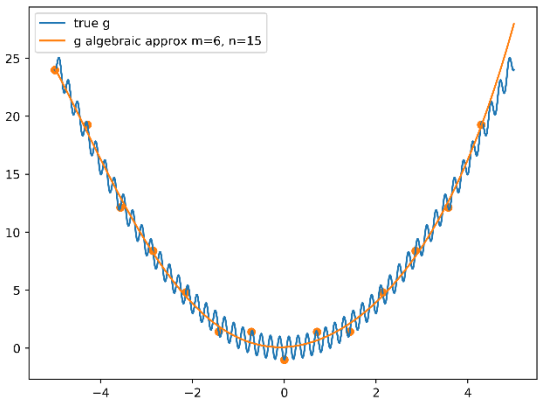

true g
g algebraic approx m=3, n=8

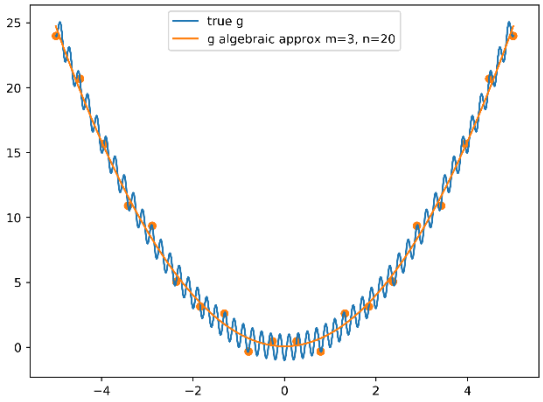

true g
g algebraic approx m=6, n=8

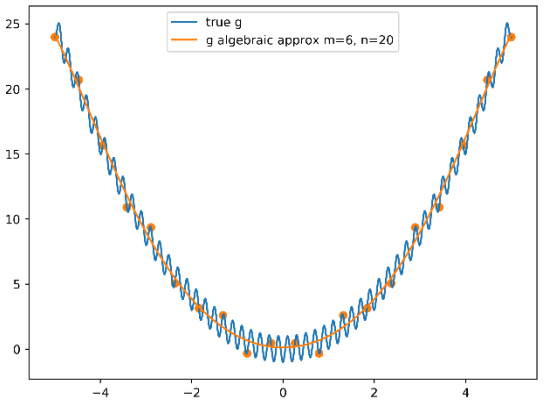

true g
g algebraic approx m=3, n=10

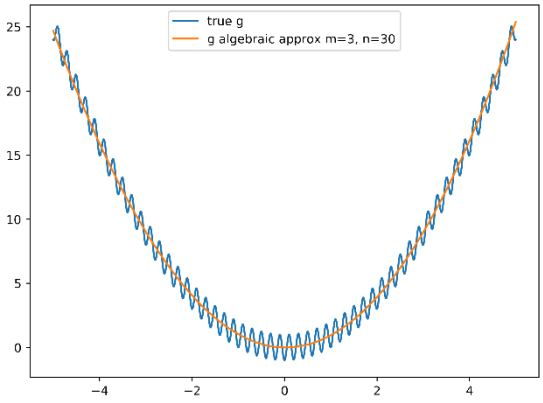

true g
g algebraic approx m=6, n=10

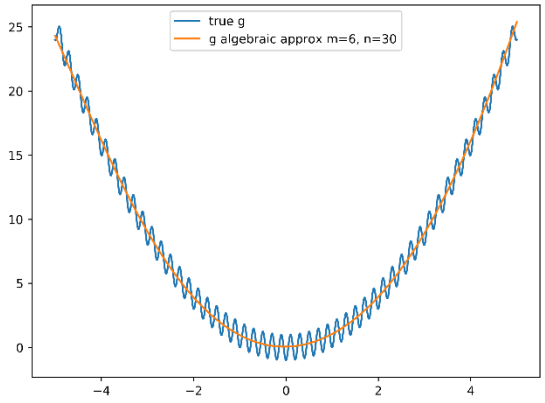

true g
g algebraic approx m=3, n=15

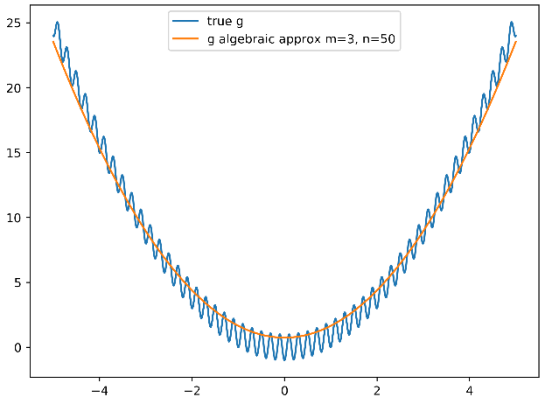

true g
g algebraic approx m=6, n=15

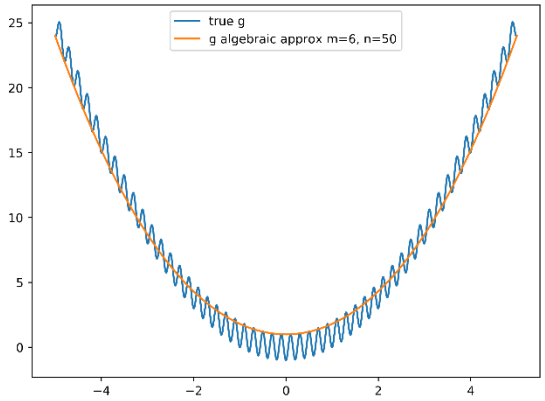

true g
g algebraic approx m=3, n=20

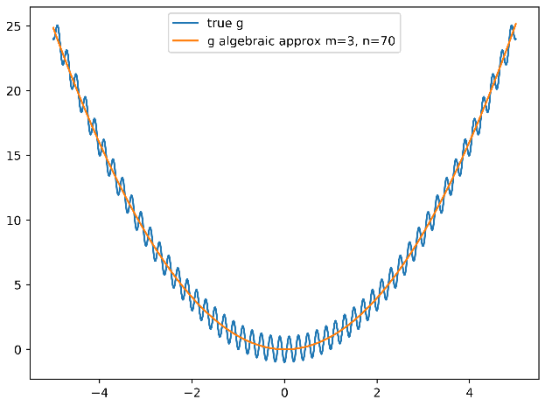

true g
g algebraic approx m=6, n=20

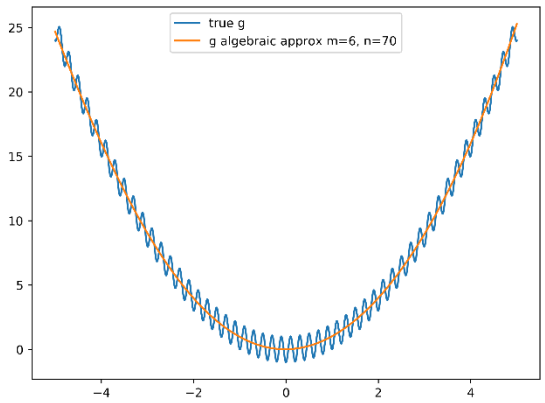


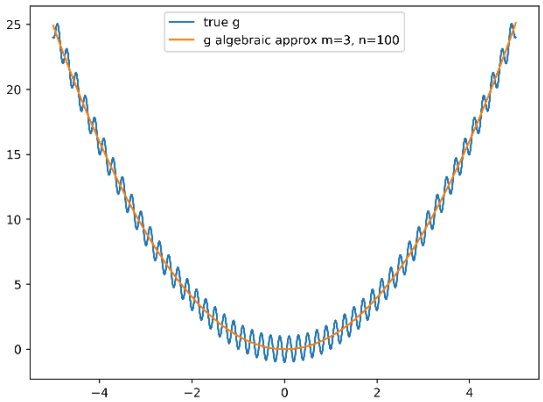


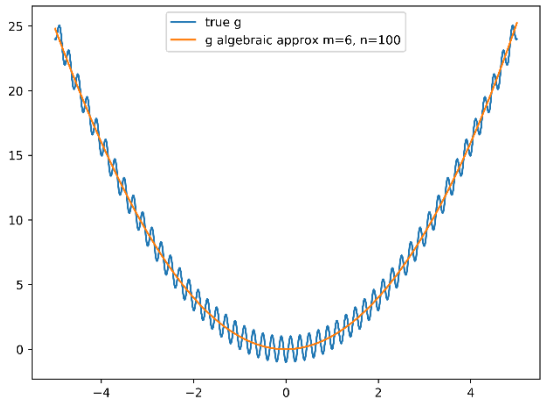








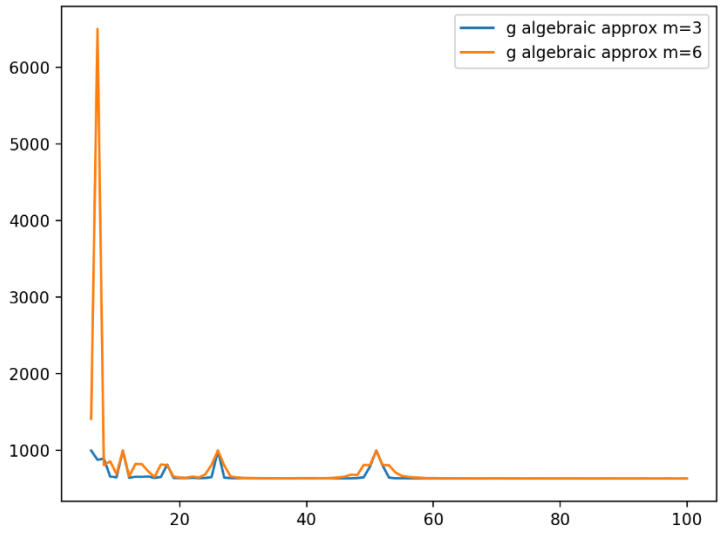




Można założyć, że dane zebrane na podstawie funkcji g dotyczą funkcji x^2, ale są obarczone błędem. Najlepiej byłoby tu do aproksymacji wykorzystać wielomian stopnia 2. W związku z tym, wielomian stopnia 3 przybliża tę funkcję lepiej niż wielomian stopnia 6. Mimo to, oba wielomiany zachowują się

tu bardzo podobnie i dobrze przybliżają funkcję. Zmieniłoby się to jednak, gdyby waga funkcji cos we wzorze funkcji g była większa, tj. rozrzut (błąd) byłby większy na niekorzyść wielomianu o stopniu 6.

Na wykresie poniżej przedstawiono błędy:



Tak jak poprzednio, błąd wielomianu stopnia 6 jest na początku dużo większy. Później jednak mniej więcej zrównuje się z błędem wielomianu stopnia 3. Warto tu zwrócić uwagę na dwa piki wykresu w okolicach $n = 26$ i $n = 50$. Ma na to wpływ rozłożenie punktów. Może się bowiem zdarzyć tak, że większość punktów wejściowych aproksymacji będzie dana z dużym błędem.

3. Aproksymacja średniokwadratowa trygonometryczna

W dalszym ciągu rozważamy aproksymację średniokwadratową, więc będziemy minimalizować sumę kwadratów błędów. W tym przypadku natomiast, szukać będziemy kombinacji liniowych funkcji trygonometrycznych:

$$1, \cos x\,, \sin x\,, \cos 2x\,, \sin 2x\,, \dots, \cos mx\,, \sin mx$$

Funkcja aproksymująca będzie miała postać:

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{m} [a_k \cos kx + b_k \sin kx]$$

Wprowadźmy oznaczenie: $L = \frac{n}{2}$, gdzie n to liczba punktów wejściowych. Zakładamy, że punkty aproksymujące są równoodległe. Funkcję f będziemy rozpatrywać w przedziale $[0,2\pi]$, punkty te przybiorą więc postać:

$$x_i = \frac{\pi i}{L}\ dla\ i = 0,1, \dots, 2L - 1$$

Wartości punktów $F(x_i)$ pozostają takie same.

Szukamy teraz minimum funkcji:

$$H(a_0, a1, b1, \dots, a_m, b_m) = \sum_{i=0}^{n-1} [F(x_i) - f(x_i)]^2$$

Obliczając pierwsze pochodne funkcji H po współczynnikach oraz wykorzystując odpowiednie tożsamości i przekształcenia otrzymujemy wyrażenia na szukane współczynniki:

$$a_k = \frac{1}{L} \sum_{i=0}^{n-1} F(x_i) \cos\left(k \frac{\pi i}{L}\right)$$

$$b_k = \frac{1}{L} \sum_{i=0}^{n-1} F(x_i) \sin\left(k \frac{\pi i}{L}\right)$$

Zaimplementowano funkcję *trygonometricLeastSquares* znajdującą współczynniki a_k i b_k funkcji aproksymującej średniokwadratowo.

```
std::pair<std::vector<double>, std::vector<double>> trygonometricLeastSquares
                                                (std::vector<Point> points, int m)
{
    m++;
    int n = points.size();
    double L = n / 2.0;
    std::vector<double> A(m, 0);
    std::vector<double> B(m, 0);

    for (int k = 0; k < m; k++)
    {
        for (int i = 0; i < n; i++)
        {
            A[k] += points[i].getY() * cos(k * M_PI * i / L);
            B[k] += points[i].getY() * sin(k * M_PI * i / L);
        }
        A[k] /= L;
        B[k] /= L;
    }

    return std::make_pair(A, B);
}
```

Funkcja ta przyjmuje zbiór punktów aproksymujących i liczbę m wyznaczającą maksymalny mnożnik w argumentach funkcji trygonometrycznych. Zwracana jest para tablic współczynników: A i B. Współczynniki te można już wykorzystać do aproksymacji. Należy jednak pamiętać o odpowiednim przeskalowaniu przedziału wzorem:

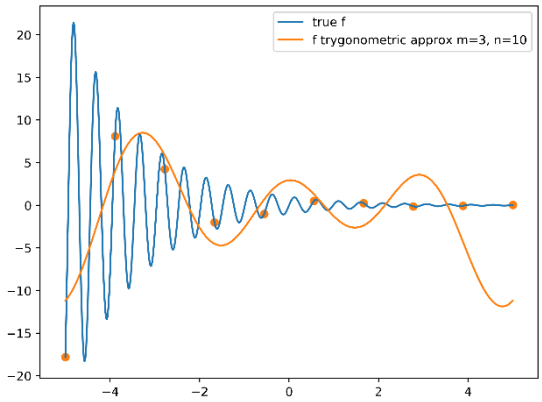
$$x_{new} = \frac{x - min}{max - min} \cdot (max_{new} - min_{new}) + min_{new}$$

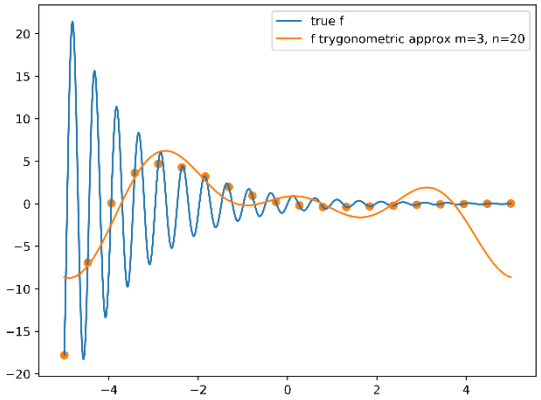
Jest to wykonywane przez funkcję *findTrygonometricPoints*:

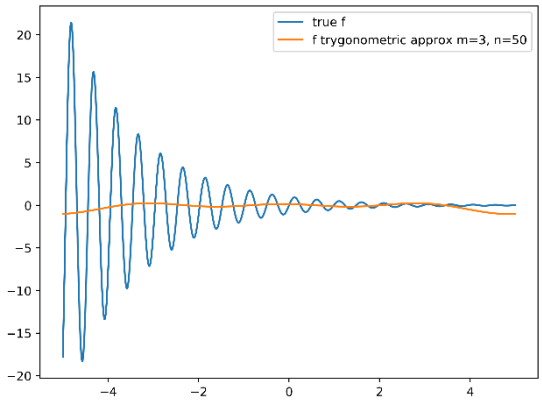
```cpp
std::vector<Point> findTrygonometricPoints(std::vector<double> X,
std::vector<double> A, std::vector<double> B, double start, double end)
{
    std::vector<Point> result;
    for (double x : X)
    {
        double y = A[0] / 2;
        for (int k = 1; k < A.size(); k++)  // Counting + transformation from
                                            // [start, end] to [0, 2*PI]
        {
            y += A[k] * cos(k * ((x - start) / (end - start) * 2 * M_PI));
            y += B[k] * sin(k * ((x - start) / (end - start) * 2 * M_PI));
        }
        result.push_back(Point(x, y));
    }
    return result;
}
```

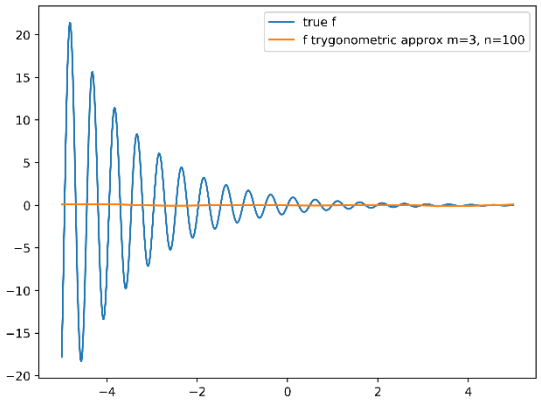
Funkcja ta przyjmuje tablicę X punktów, które chcemy aproksymować, tablice A i B współczynników oraz granice przedziału *start* i *end*. Funkcja zwraca tablicę punktów aproksymowanych.

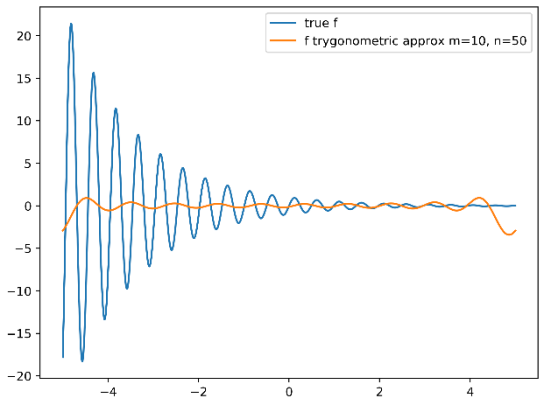
Funkcje testowe będę przybliżał wykorzystując różne liczby m oraz różne liczby punktów n. Wykresy dla funkcji testowej f przedstawiono poniżej.

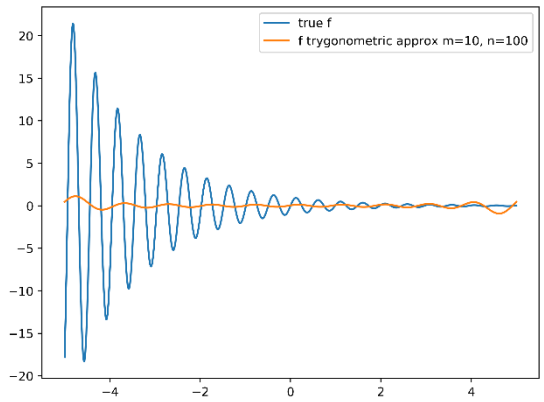


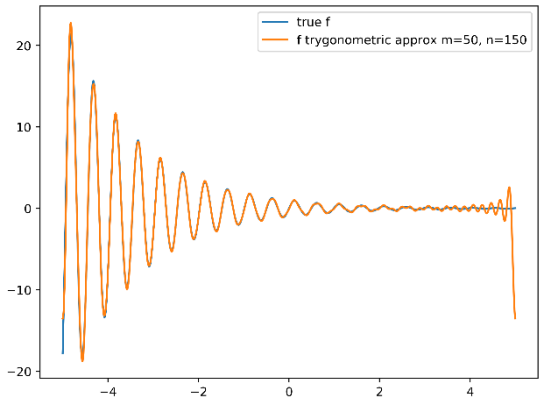


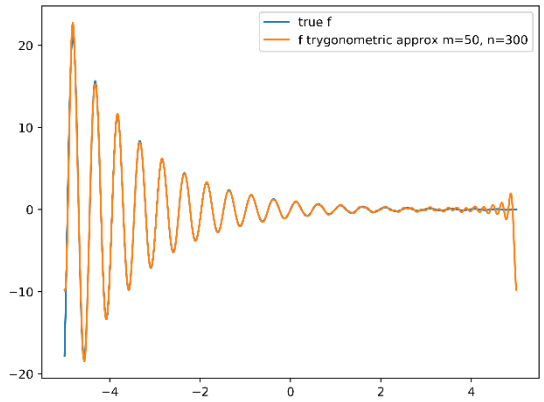


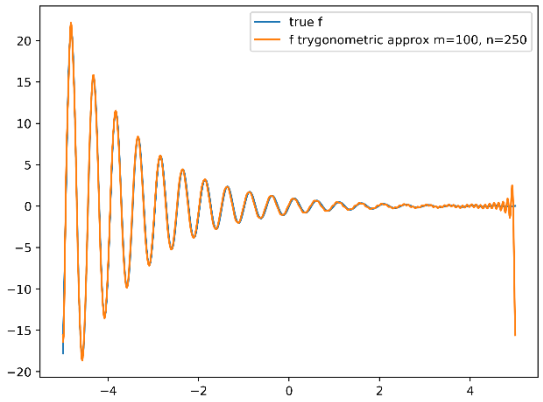


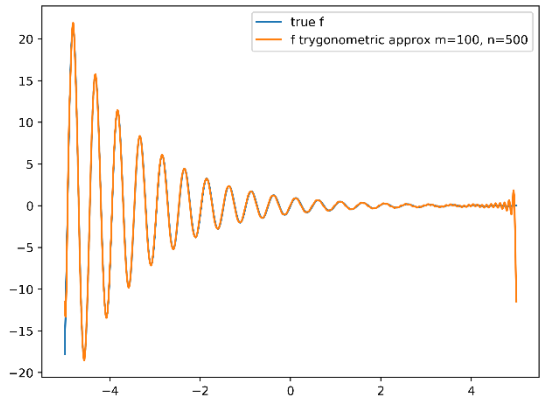




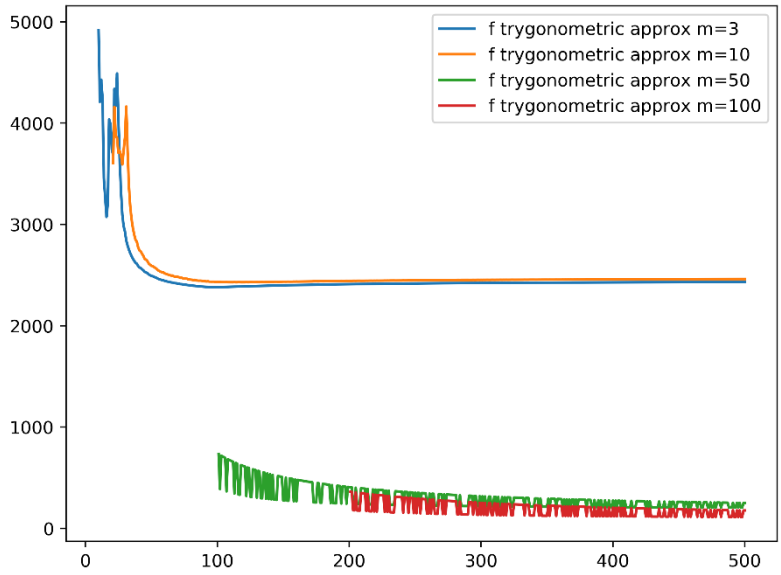






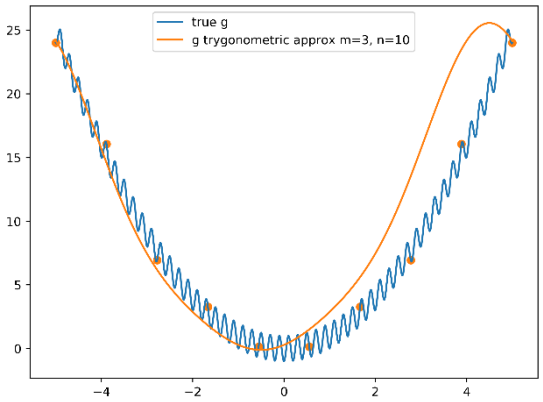


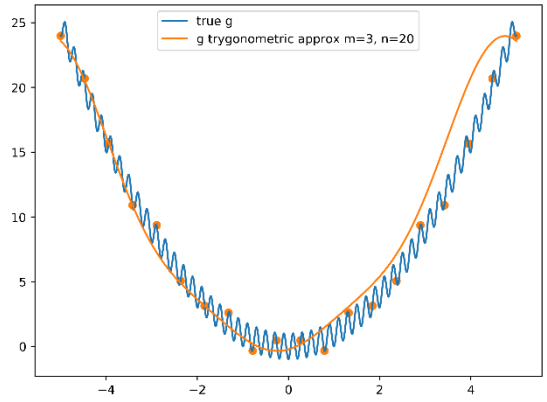
Jak widać, tak jak w przypadku poprzedniej metody, dla małych liczb punktów najistotniejszym czynnikiem jest ich rozmieszczenie. Funkcje dla małych liczb m zachowują się podobnie do wielomianów algebraicznych. Dla $m = 10$ widać już jednak wyraźnie wpływ funkcji trygonometrycznych. Dla większych liczb m, funkcja przybliżana jest coraz lepiej, jednak nawet w „najmocniejszej" konfiguracji, tj. $m = 100, n = 500$, na prawym krańcu przedziału pojawiają się spore błędy. Mimo tego, to właśnie tutaj przybliżenie jest najlepsze. Można to zaobserwować na poniższym wykresie rozmiaru błędu od liczby punktów:

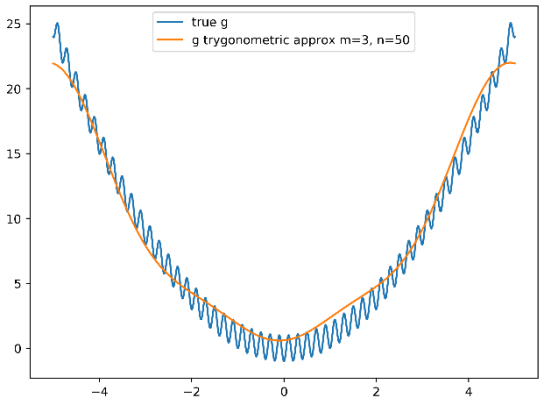


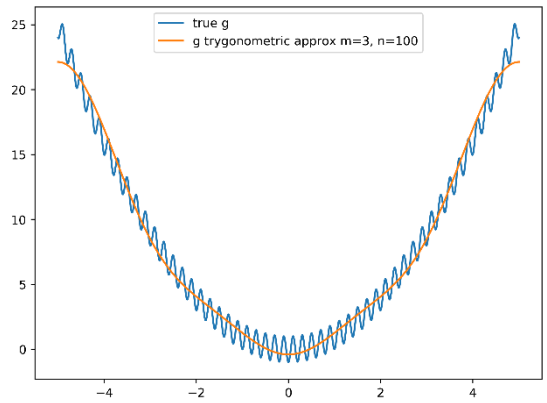
Funkcje dla $m = 3$ i $m = 10$ zachowują się podobnie. Dużo lepsze są funkcje dla $m = 50$ i $m = 100$. Rozrzuty na powyższym wykresie spowodowane są rozmieszczeniem punktów. Wystarcza więc $m = 50$ i $n = 101$, żeby funkcja została przybliżona w miarę dokładnie. Dalsze dokładnie punktów aproksymujących zmniejsza błąd już nieznacznie.

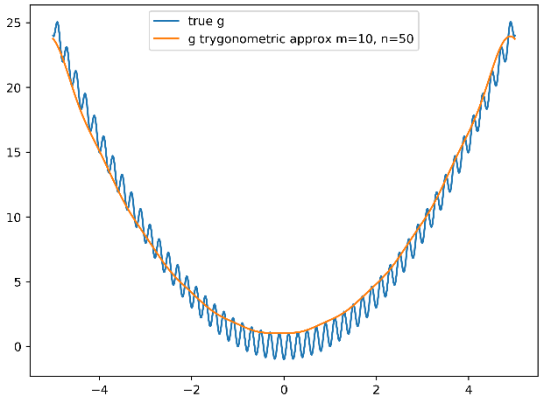
Analogiczne wykresy dla funkcji g przedstawiono poniżej.

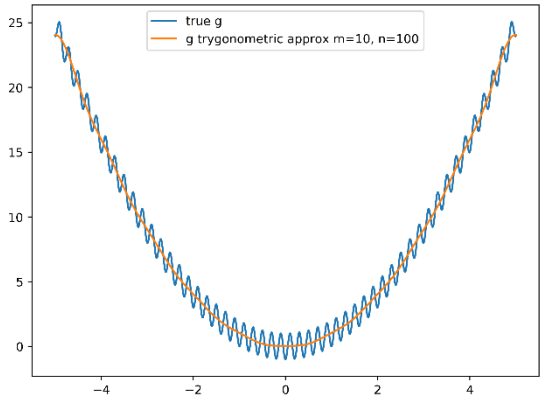


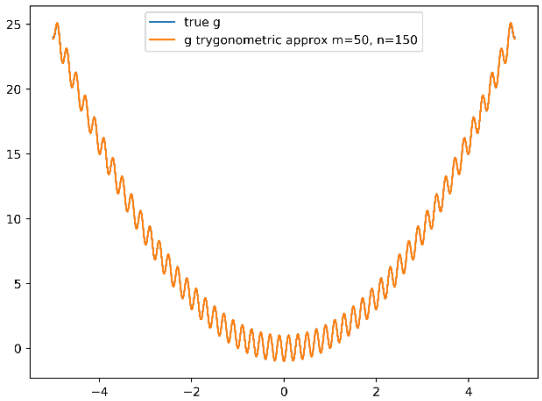


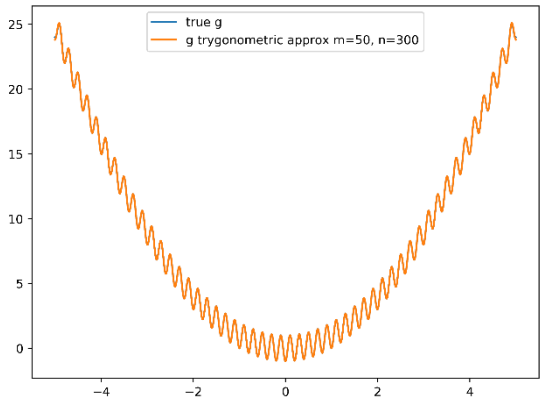


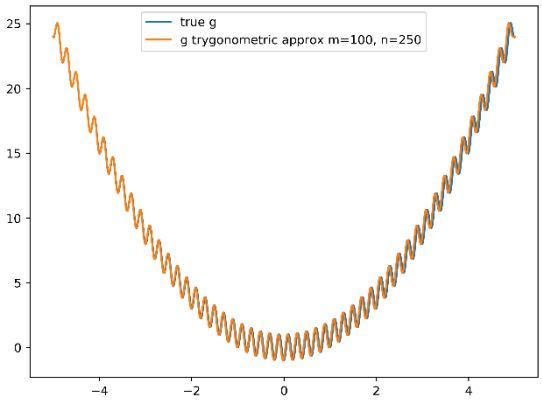


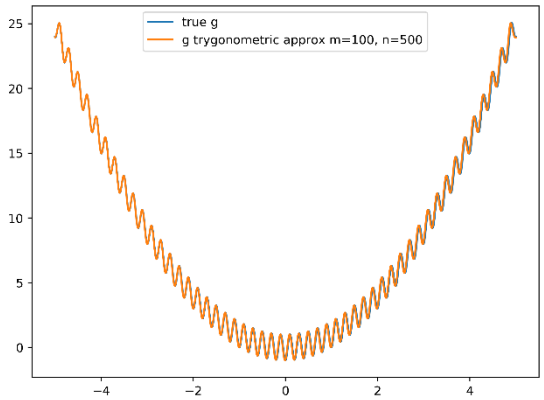




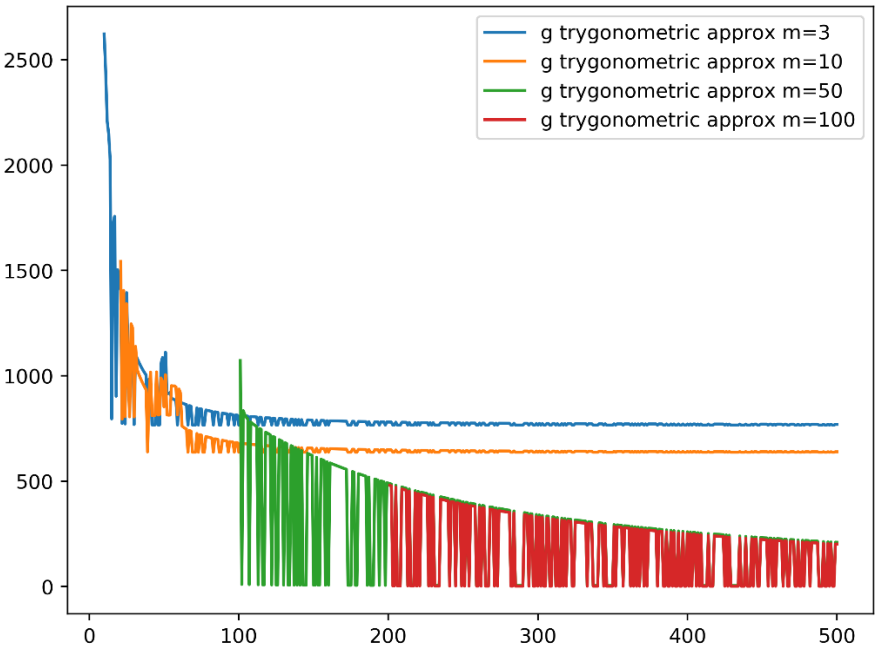








Dla $m = 3$ i $m = 10$ przybliżany jest tylko składnik funkcji g: x^2. Dla $m = 50$ natomiast, całkowity kształt funkcji g jest już przybliżany bardzo dokładnie. Można to zobaczyć na wykresie błędów poniżej.



Jak widać, wykresy błędów dla $m = 50$ i $m = 100$ wyglądają bardzo podobnie. Tak jak wcześniej, duże wahania wykresów są spowodowane rozmieszczeniem punktów.