Laboratorium 5: Aproksymacja

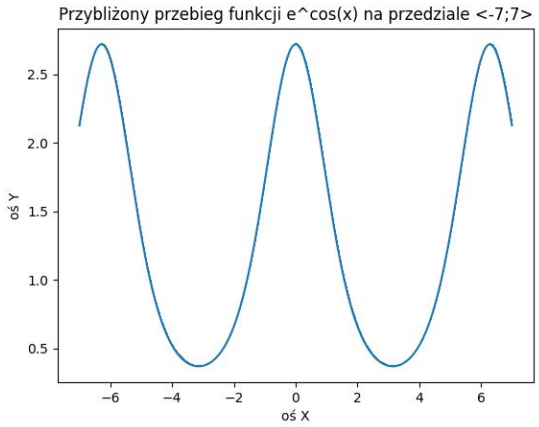
Metody Obliczeniowe w Nauce i Technice

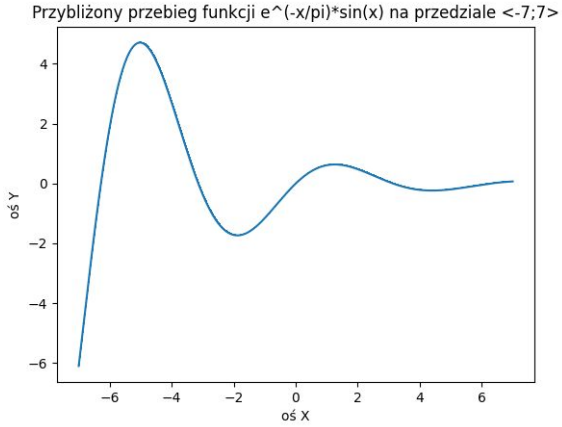
Wiktor Tarsa

1. Funkcje testowe

Wybrane przeze mnie funkcje testowe to: $e^{cos(x)}$ i $sin(x) * e^{\frac{-x}{\pi}}$. Zaimplementowałem te funkcje w programie. Następnie korzystając z funkcji utworzonej w laboratorium 4 wyznaczyłem wartości obu funkcji w 10001 równoodległych punktach na przedziale <-7; 7>. Na tym przedziale będę wykonywał aproksymację.

```
double f1(double x){
   return exp(cos(x));
}

double f2(double x){
   return exp(-x/M_PI) * sin(x);
}
```





2. Aproksymacja średniokwadratowa wielomianami algebraicznymi

W aproksymacji średniokwadratowej wielomianami algebraicznymi naszym zadaniem jest znalezienie wielomianu uogólnionego którego wzór ogólny to suma iloczynów współczynników i poszczególnych funkcji bazowych. Na początku dostajemy lub przyjmujemy określony układ funkcji bazowych, dlatego zadanie sprowadza się do wyznaczenia wartości współczynników. W wykonywanej aproksymacji przyjmuję, że każdy punkt ma tą samą wagę (równą 1).
Aby wyznaczyć te współczynniki skorzystałem z wyprowadzenia z wykładu:

$$\begin{pmatrix} \sum w_i & \sum w_i x_i & \sum w_i x_i^2 & \dots & \sum w_i x_i^m \\ \sum w_i x_i & \sum w_i x_i^2 & \sum w_i x_i^3 & \dots & \sum w_i x_i^{m+1} \\ . & . & . & . & . \\ \sum w_i x_i^m & \sum w_i x_i^{m+1} & \sum w_i x_i^{m+2} & \dots & \sum w_i x_i^{2m} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{pmatrix} =$$

$$= \begin{pmatrix} \sum w_i F_i \\ \sum w_i F_i x_i \\ \vdots \\ \sum w_i F_i x_i^m \end{pmatrix}$$

$$\underline{G \cdot A = B}$$

Jako układ funkcji bazowych przyjąłem jednomiany postaci x^k, gdzie k = 0, 1, …, m.
Węzły aproksymacji wyznaczyłem korzystając z funkcji napisanej w laboratorium o interpolacji - wyznaczyłem określoną ilość węzłów równoodległych na przedziale <-7; 7>.
Aby wyznaczyć macierz współczynników napisałem kilka pomocniczych funkcji:

calculateSum

Oblicza wartość pojedynczych wyrazów macierzy.

```
double calculateSum(int power, std::vector<point> points){
   double result = 0.0;
   for(point p: points){
       result += pow(p.x, power);
   }
   return result;
}

double calculateSum(int power, std::vector<point> points, double
func(double)){
   double result = 0.0;
   for(point p: points){
       result += pow(p.x, power)*func(p.x);
   }
   return result;
}
```

Do tworzenia macierzy i rozwiązywania układu równań użyłem klasy AGHMatrix zaimplementowanej na laboratorium 2. Poniższe funkcje korzystają z tej klasy:

createMainMatrix

Tworzy macierz zawierającą układ równań.

```
AGHMatrix<double> createMainMatrix(int m, std::vector<point> points, double
func(double)){ // n = number of points, int m = number of functions
   std::vector<std::vector<double>> mat;
   std::vector<double> row;
   row.resize(m+2, 0);
   mat.resize(m+1, row);

   //first row
   for(int i = 0; i <= m; i++){
       mat[0][i] = calculateSum(i, points);
   }

   //remaining rows
   for(int i = 1; i <= m; i++){
       for(int j = 0; j <= m; j++){
           if(j != m) mat[i][j] = mat[i-1][j+1];
```

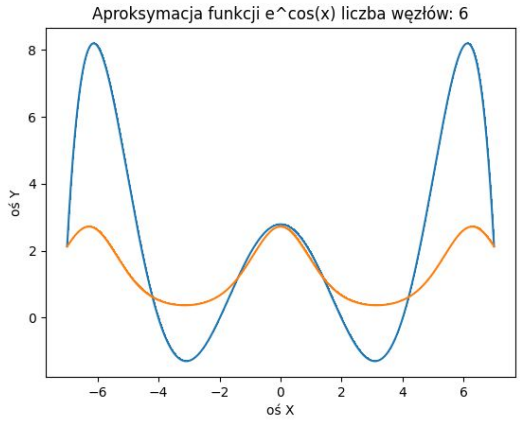
```
            else mat[i][j] = calculateSum(i+j, points);
        }
    }

    //last column (results)
    for(int i = 0; i <= m; i++){
        mat[i][m+1] = calculateSum(i, points, func);
    }

    AGHMatrix <double> result_matrix(mat);
    return result_matrix;
}
```

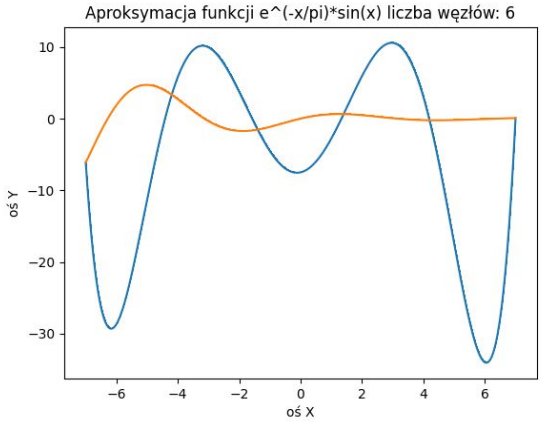
beginApproximation

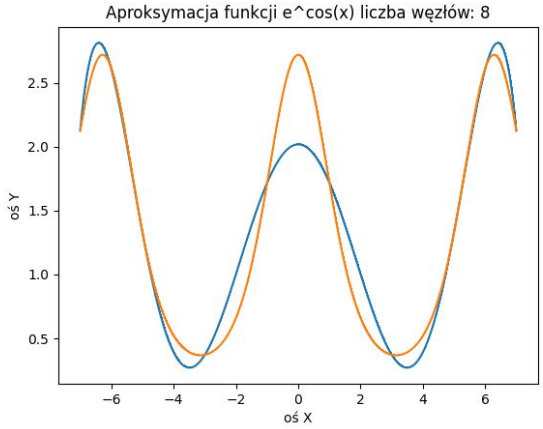
Wykonuje aproksymację dla określonej funkcji. Zwraca wektor zawierający wartości wyznaczonej funkcji aproksymującej w 10001 punktów na przedziale <-7; 7>.

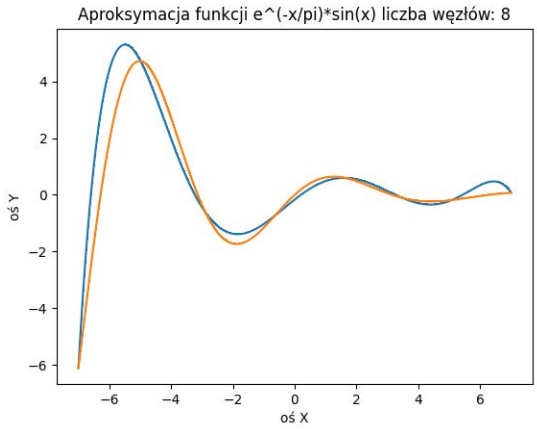
```
std::vector<point> beginApproximation(int n, int m, double func(double)){
    std::vector<point> points = getValueInPoints(func, n);
    AGHMatrix<double> mat = createMainMatrix(m, points, func);
    AGHMatrix<double> result = mat.gauss_elimination();
    std::vector<point> values;
    double interval = 14.0/(10000);
    for(int i = 0; i < 10001; i++){
        point point;
        point.x = interval*i-7;
        double y_val = 0.0;
        for(int j = 0; j < m+1; j++){
            y_val += result.matrix[j][0] * pow(point.x, j);
        }
        point.y = y_val;
        values.push_back(point);
    }
    return values;
}
```

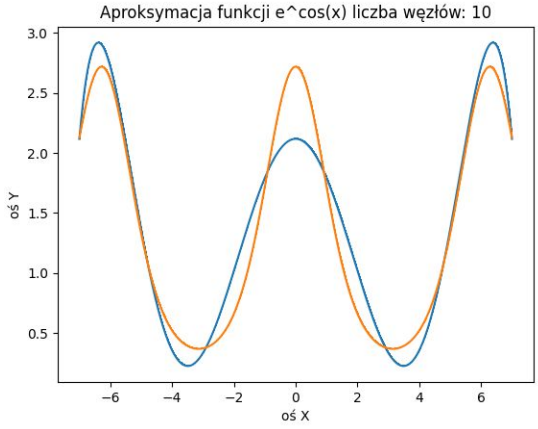
Aproksymację wykonałem dla układu sześciu funkcji bazowych i dla różnej liczby węzłów. Ćwiczenie wykonałem na przedziale <-7; 7> dla obu funkcji testowych. Poniższe wykresy przedstawiają otrzymane wyniki (kolorem niebieskim została zaznaczona funkcja aproksymująca, a kolorem żółtym - aproksymowana).

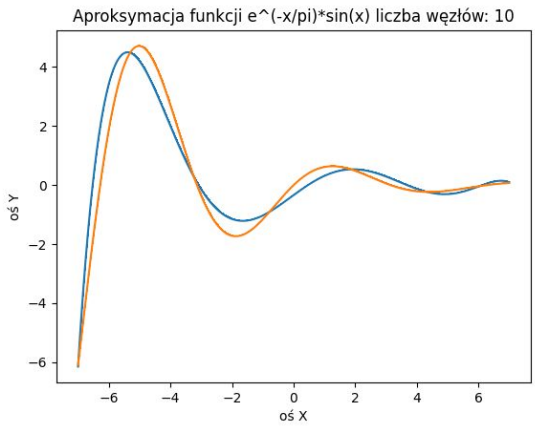

Aproksymacja funkcji e^cos(x) liczba węzłów: 6
oś Y
oś X

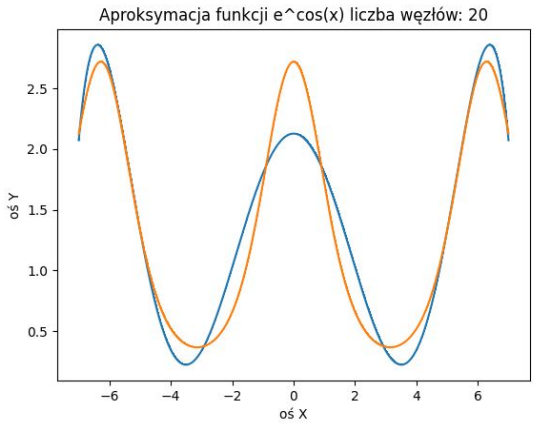

Aproksymacja funkcji e^(-x/pi)*sin(x) liczba węzłów: 6
oś Y
oś X

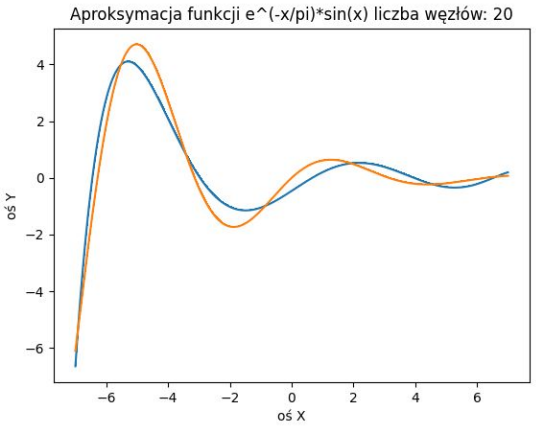

Aproksymacja funkcji e^cos(x) liczba węzłów: 8
oś Y
oś X


Aproksymacja funkcji e^(-x/pi)*sin(x) liczba węzłów: 8
oś Y
oś X


Aproksymacja funkcji e^cos(x) liczba węzłów: 10
oś Y
oś X


Aproksymacja funkcji e^(-x/pi)*sin(x) liczba węzłów: 10
oś Y
oś X





Oszacowanie błędów przybliżenia

Napisałem funkcję, która sumuje wartości bezwzględne różnic pomiędzy wartością funkcji aproksymowanej a aproksymującej w wyznaczonych punktach.

liczba węzłów	e^cos(x)	e^(-x/pi) * sin(x)
6	1.80E+04	1.15E+05
8	1.64E+03	4.31E+03
10	1.84E+03	3.82E+03
20	1.73E+03	3.86E+03
30	1.69E+03	3.70E+03
40	1.68E+03	3.59E+03
60	1.67E+03	3.47E+03
80	1.67E+03	3.40E+03
100	1.67E+03	3.37E+03

Wraz ze wzrostem liczby węzłów maleje wartość błędu aproksymacji. Jednak dla liczby węzłów większej od 20 wartość błędu maleje tak nieznacznie, że ciężko jest zauważyć różnice w wykresach.

3. Aproksymacja średniokwadratowa trygonometryczna

W tym punkcie będę badał funkcje testowe na przedziale <0; 2pi>.
Aproksymacja średniokwadratowa trygonometryczna to aproksymacja której układ funkcji bazowych to 1, sin(x), cos(x), sin(2x), cos(2x), … . W tym przypadku szukamy funkcji postaci:

$$F(x) = \frac{1}{2}a_0 + \sum_{j=1}^{n}(a_j cos(jx) + b_j sin(jx))$$

Jest to aproksymacja średniokwadratowa, dlatego powyższa funkcja musi spełniać warunek:

$$\sum_{i=0}^{2L-1}[f(x_i) - F(x_i)]^2 = min$$

Korzystając z powyższego wzoru i wzoru funkcji możemy wyznaczyć wzory na współczynniki aj i bj:

$$a_j = \frac{1}{L}\sum_{i=0}^{2L-1} f(x_i)cos(jx_i) \qquad b_j = \frac{1}{L}\sum_{i=0}^{2L-1} f(x_i)sin(jx_i)$$
$$= \frac{1}{L}\sum_{i=0}^{2L-1} f(x_i)cos\frac{\pi ij}{L} \qquad = \frac{1}{L}\sum_{i=0}^{2L-1} f(x_i)sin\frac{\pi ij}{L}$$

W powyższych oznaczeniach L to liczba węzłów, a n - liczba funkcji bazowych. Podobnie jak we wcześniejszym przypadku - musi zachodzić warunek n < L by istniała dokładnie jedna funkcja spełniająca warunki problemu.

Powyższe wzory pochodzą ze strony http://home.agh.edu.pl/~chwiej/mn/aproksymacja.pdf . W slajdach z wykładu niestety nie znalazłem żadnych informacji na temat aproksymacji średniokwadratowej trygonometrycznej.

Implementacja

Utworzyłem kilka funkcji pomocniczych:

getTrigonometricPoints

Wyznacza węzły aproksymacji na przedziale <0;2pi>

```
std::vector<point> getTrigonometricPoints(int L, double func(double)){
   std::vector<point> points;
   for(int i = 0; i < 2*L; i++){
       point p;
       p.x = i*M_PI/L;
       p.y = func(p.x);
       points.push_back(p);
   }
   return points;
}
```

getAj i getBj

Wyznaczają wartości współczynników aj i bj.

```
double getAj(int L, int j, std::vector<point> points){
   double a = 0.0;
   for(int i = 0; i < 2*L; i++){
       a += points[i].y*cos(M_PI*i*j/L);
   }
   a /= L;
   return a;
}

double getBj(int L, int j, std::vector<point> points){
   double b = 0.0;
   for(int i = 0; i < 2*L; i++){
       b += points[i].y*sin(M_PI*i*j/L);
   }
   b /= L;
   return b;
}
```

getFunctionValue

Wyznacza wartość funkcji interpolującej o określonej współrzędnej.

```
double getFunctionValue(int L, double x, double func(double), int n){
   std::vector<point> points = getTrigonometricPoints(L, func);
   double result = getAj(L, 0, points)/2;
   for(int j = 1; j <= n; j++){
       result += getAj(L, j, points)*cos(j*x);
       result += getBj(L, j, points)*sin(j*x);
   }
   return result;
}
```

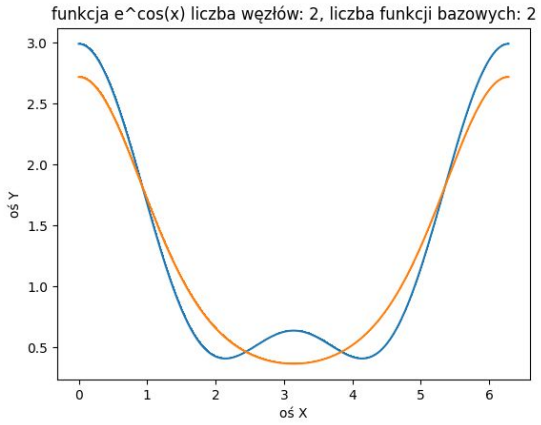
beginApproximationTrigonometric

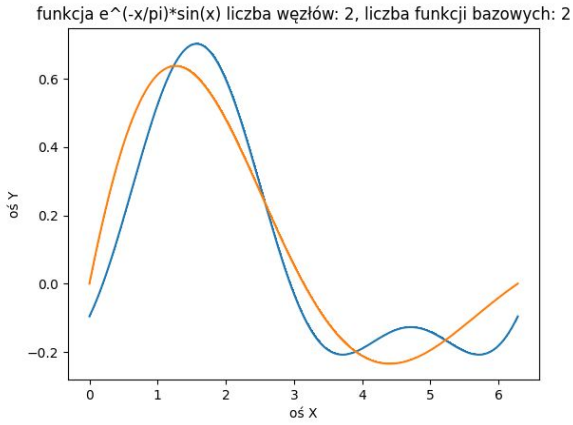
Wyznacza wartość funkcji aproksymującej w 10001 punktów na przedziale <0, 2pi>.

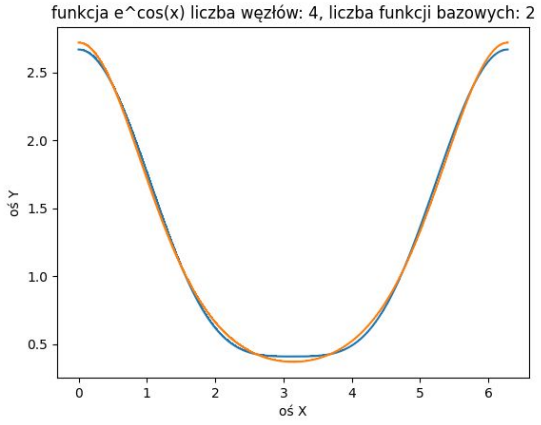
```
std::vector<point> beginApproximationTrigonometric(int n, int L, double
func(double)){
   std::vector<point> values;
   double interval = 2*M_PI/(10000);
   for(int i = 0; i < 10001; i++){
       point point;
       point.x = interval*i;
       point.y = getFunctionValue(L, point.x, func, n);
       values.push_back(point);
   }
```

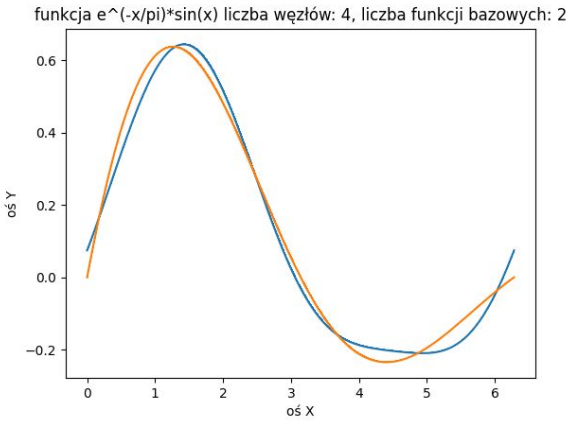
```
    return values;
}
```

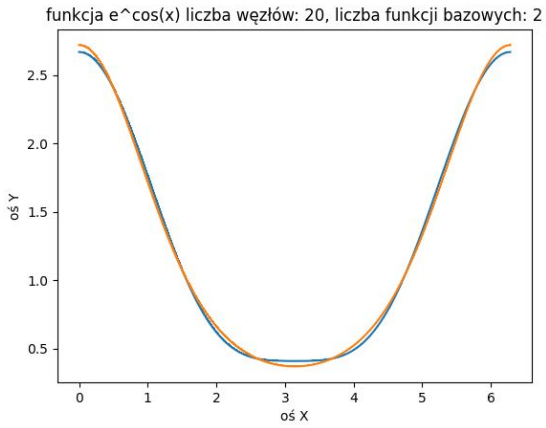
Tym razem aproksymację wykonałem dla różnych ilości funkcji bazowych. Ćwiczenie wykonałem na przedziale <0; 2pi> dla obu funkcji testowych. Poniższe wykresy przedstawiają otrzymane wyniki (kolorem niebieskim została zaznaczona funkcja aproksymująca, a kolorem żółtym - aproksymowana).

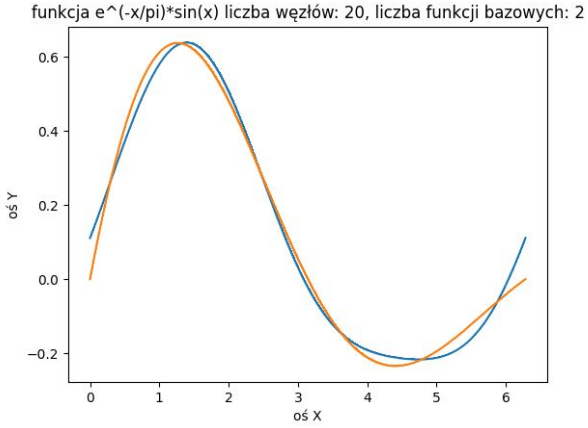


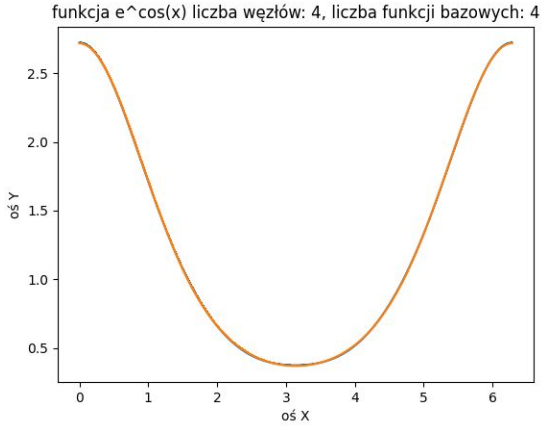


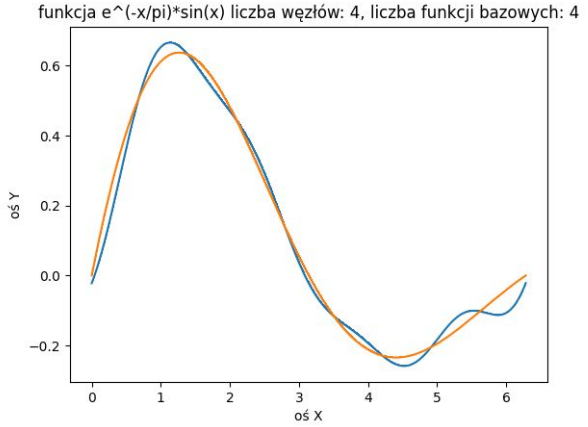


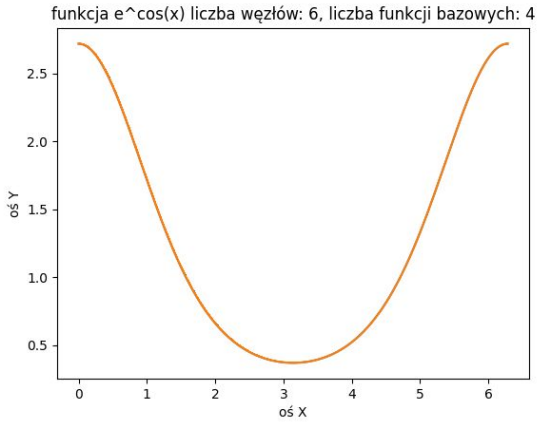


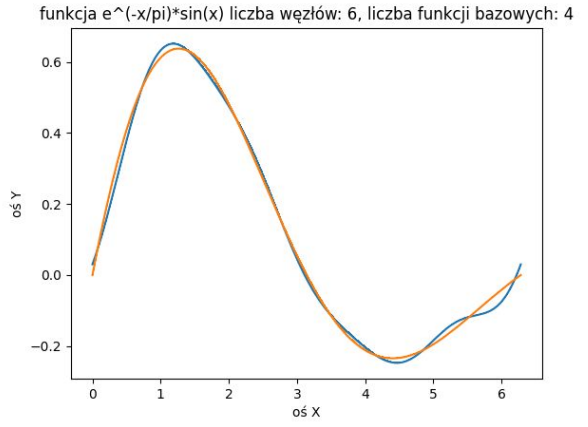

funkcja e^cos(x) liczba węzłów: 20, liczba funkcji bazowych: 2
oś Y
oś X

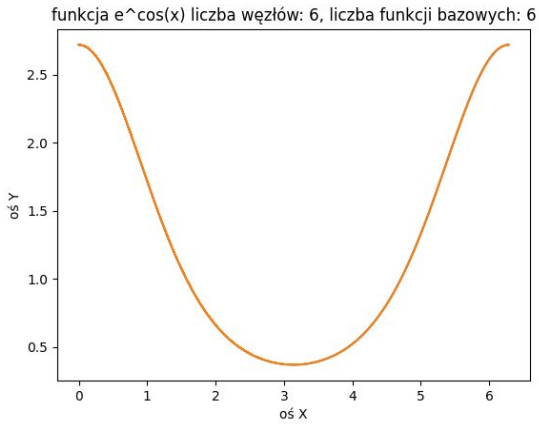

funkcja e^(-x/pi)*sin(x) liczba węzłów: 20, liczba funkcji bazowych: 2
oś Y
oś X

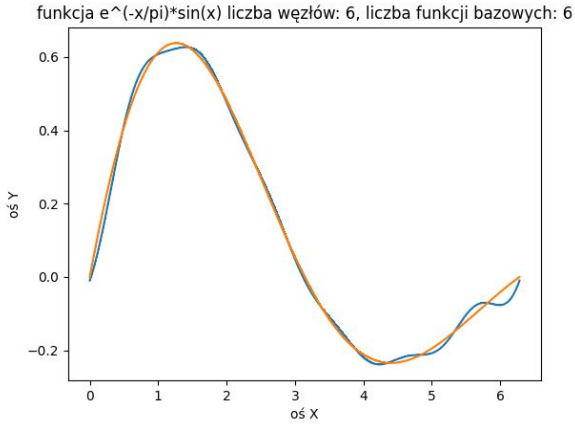

funkcja e^cos(x) liczba węzłów: 4, liczba funkcji bazowych: 4
oś Y
oś X


funkcja e^(-x/pi)*sin(x) liczba węzłów: 4, liczba funkcji bazowych: 4
oś Y
oś X


funkcja e^cos(x) liczba węzłów: 6, liczba funkcji bazowych: 4
oś Y
oś X


funkcja e^(-x/pi)*sin(x) liczba węzłów: 6, liczba funkcji bazowych: 4
oś Y
oś X





Łatwo dostrzec, że ten sposób aproksymacji jest dużo dokładniejszy od aproksymacji wielomianami. Zarówno zwiększenie liczby węzłów, jak i zwiększenie liczby funkcji bazowych polepsza dokładność aproksymacji.

Oszacowanie błędów przybliżenia

Wyznaczyłem błąd w sposób analogiczny do sposobu w poprzednim punkcie. Wyniki przedstawia poniższa tabela.

liczba węzłów	l. funkcji bazowych	e^cos(x)	e^(-x/pi) * sin(x)
2	2	1780	899
4	2	283	293
6	2	283	255
20-100	2	283	237-235
4	4	35.2	250
6	4	3.46	127
10-100	4	3.46	85.9-96.4
7-100	6	0.0204	45.7-122

Otrzymane wyniki są o kilka rzędów wielkości mniejsze od wyników otrzymanych w punkcie poprzednim. Metoda aproksymacji średniokwadratowej trygonometrycznej jest bardzo dokładna. Otrzymane wielkości błędów są tak małe, że w niektórych przypadkach ciężko dostrzec różnicę na wykresie pomiędzy funkcją aproksymowaną a aproksymującą.
Wraz ze wzrostem liczby węzłów i liczby funkcji bazowych maleje wartość błędu, czyli rośnie dokładność.