Krzosa Karol **grupa 22/laboratoryjna 03, Informatyka.**

Sprawozdanie z zajęć 11 z przedmiotu "Metody Obliczeniowe", prowadzący: dr.hab.inż. L. Bieniasz.

1. Zadanie:

Zagadnienie z warunkiem początkowym i brzegowym obejmuje:

równanie różniczkowe cząstkowe $\frac{\partial U(x,t)}{\partial t} = D\left[\frac{\partial^2 U(x,t)}{\partial x^2} + \pi^2 \sin(\pi x)\right]$, określone dla współrzędnej przestrzennej $x \in [0, 1]$ oraz czasu $t \in [0, t_{max}]$,

warunek początkowy $U(x,0) = 0$, oraz

warunki brzegowe $U(0,t) = 0$, $U(1,t) = 0$.

Zagadnienie to może opisywać powstanie stanu ustalonego dla stężenia substancji o współczynniku dyfuzji D, w membranie o grubości 1 i przenikalnych ściankach, w wyniku ucieczki substancji z membrany wskutek transportu dyfuzyjnego, oraz powstawania tej substancji wewnątrz membrany.

Rozwiązanie analityczne tego zagadnienia ma postać: $U(x,t) = \left[1 - \exp\left(-\pi^2 Dt\right)\right]\sin(\pi x)$.

Należy rozwiązać to zagadnienie stosując zaznaczoną niżej kombinację algorytmów numerycznych oraz podane wartości parametrów. Należy przyjąć ustaloną wartość $\lambda = D\,\delta t/h^2$, możliwie najbliższą $\lambda = 0.4$ dla metody bezpośredniej lub $\lambda = 1$ dla metod pośrednich (uwaga na ograniczenia stabilności numerycznej!). Rozwiązania numeryczne należy porównać z analitycznymi i wyznaczyć błędy bezwzględne rozwiązań numerycznych. Jeżeli poniżej zaznaczono dwa alternatywne algorytmy, to wówczas w programie należy zrealizować oba, a uzyskane wyniki porównać.

Do zaliczenia projektu należy wykonać:

(1) Wykresy zależności maksymalnej wartości bezwzględnej błędu obserwowanej dla t_{max}, w funkcji kroku przestrzennego h (najlepiej w skali logarytmicznej, o ile to możliwe). Należy sprawdzić, czy zależność jest zgodna z teoretycznym rzędem dokładności i wyjaśnić ewentualne niezgodności. Do dalszych wykresów należy dobrać krok czasowy (i przestrzenny) tak, aby uzyskać możliwie jak najlepszą dokładność rozwiązania w czasie obliczeń nie przekraczającym około jednej minuty, dla najszybszego z rozważanych wariantów obliczeń. Wyniki numeryczne oraz rozwiązania analityczne i błędy odpowiadające tej sytuacji należy zapisać w zbiorze, w postaci sformatowanej umożliwiającej przeglądanie wyników.

(2) Wykresy rozwiązań numerycznych i analitycznych dla kilku wybranych wartości czasu t z całego przedziału t (rozwiązania numeryczne punktami, rozwiązania analityczne linią ciągłą).

(3) Wykresy zależności maksymalnej wartości bezwzględnej błędu w funkcji czasu t. Należy wyjaśnić ewentualnie obserwowane zmiany błędu w czasie.

Dyskretyzacja:

-Klasyczna metoda Bezpośrednia

-Metoda pośrednia Cranka-Nicolson

Parametry:

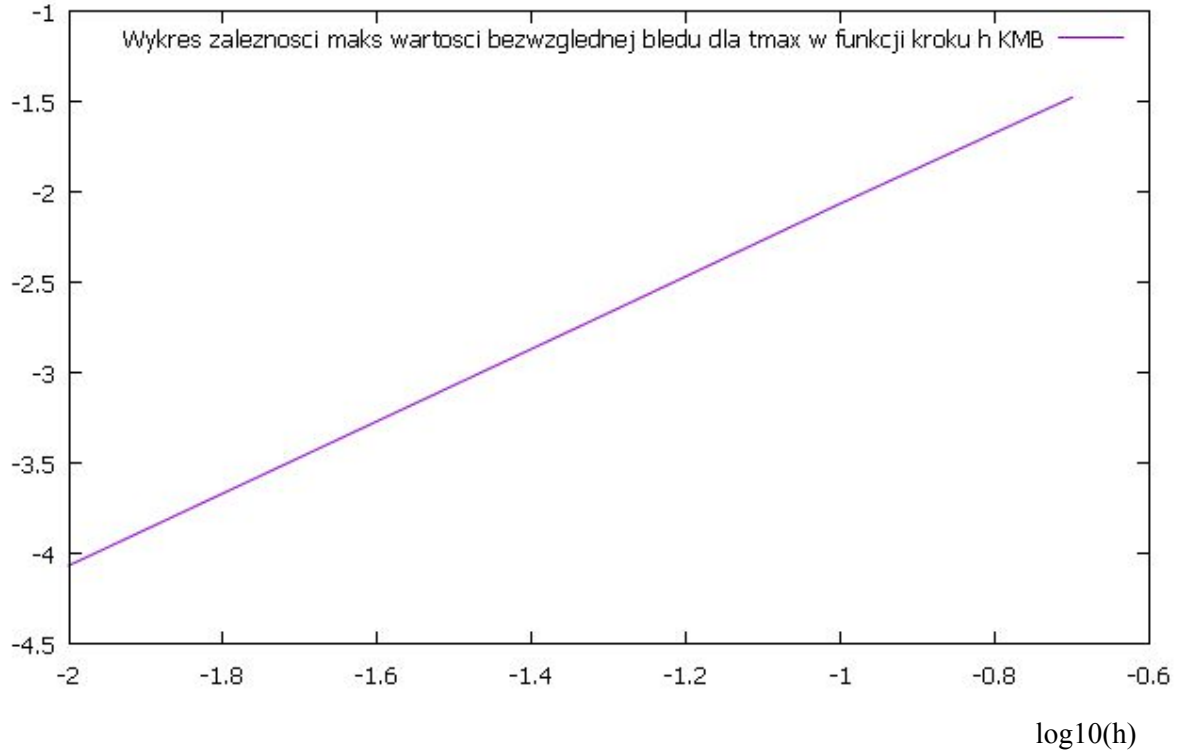
-tmax = 0.5

-D = 1

2. Wykresy

a) Wykres zależności maksymalnej wartości bezwzględnej błędu obserwowanej dla tmax w funkcji kroku przestrzennego h dla klasycznej metody bezpośredniej.

log10(blad)



log10(h)

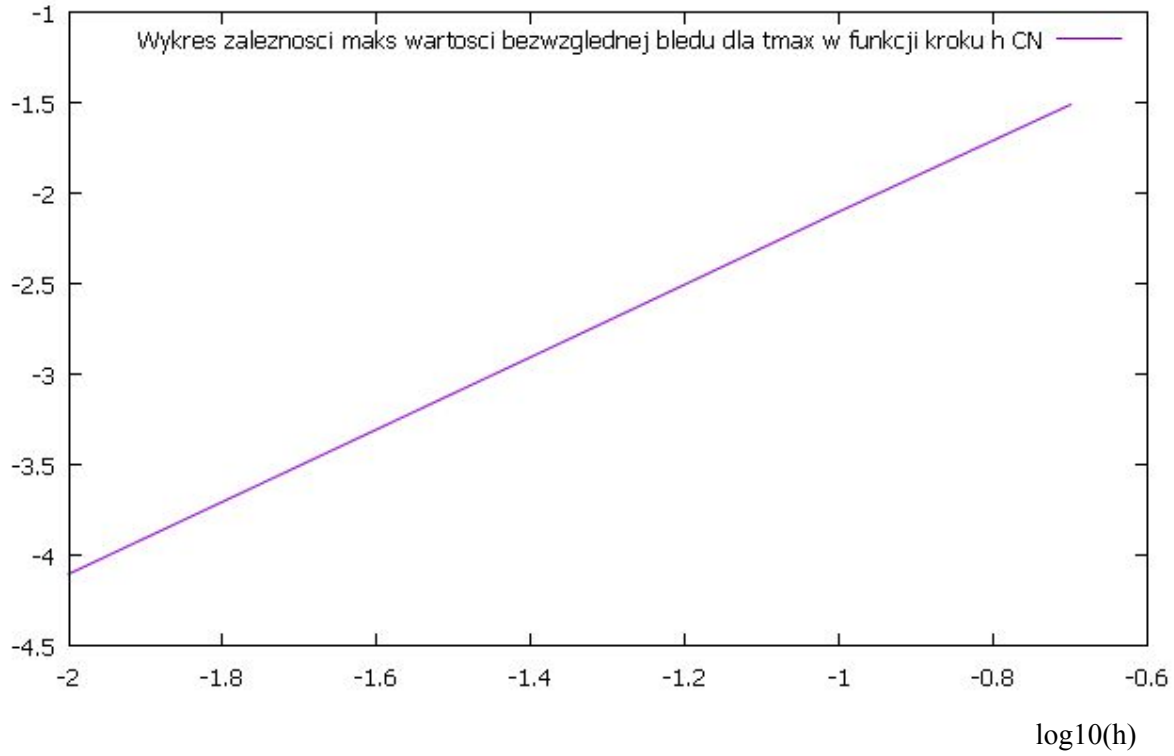
Obliczanie rzędu dla klasycznej metody bezpośredniej::

$2,55/1,3 =$ **1,96**

Rząd teoretyczny: **2**

a)Wykres zależności maksymalnej wartości bezwzględnej błędu obserwowanej dla tmax w funkcji kroku przestrzennego h dla metody pośredniej Cranka-Nicolson.

log10(blad)



Obliczanie rzędu dla metody pośredniej Cranka-Nicolson:

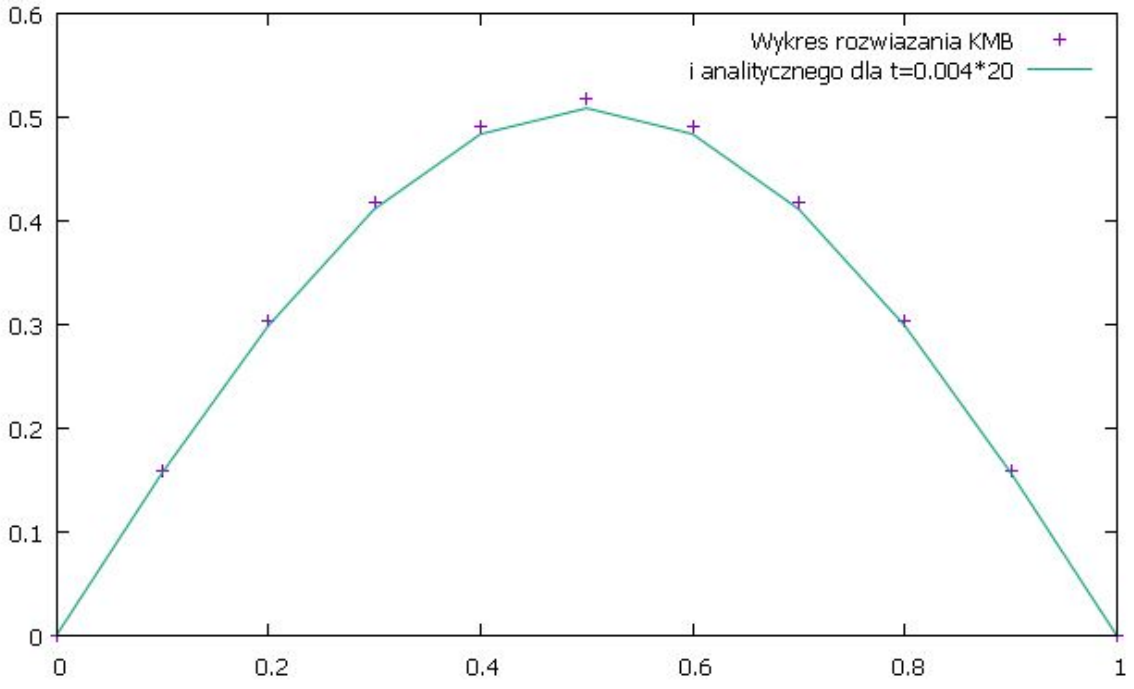
$2,6/1,3 = \mathbf{2}$

Rząd teoretyczny: **2**

b)Wykresy rozwiązań numerycznych i analitycznych dla kilku wybranych wartości czasu t z całego przedziału t. Klasyczna metoda bezpośrednia.

dla **t = 0.08**

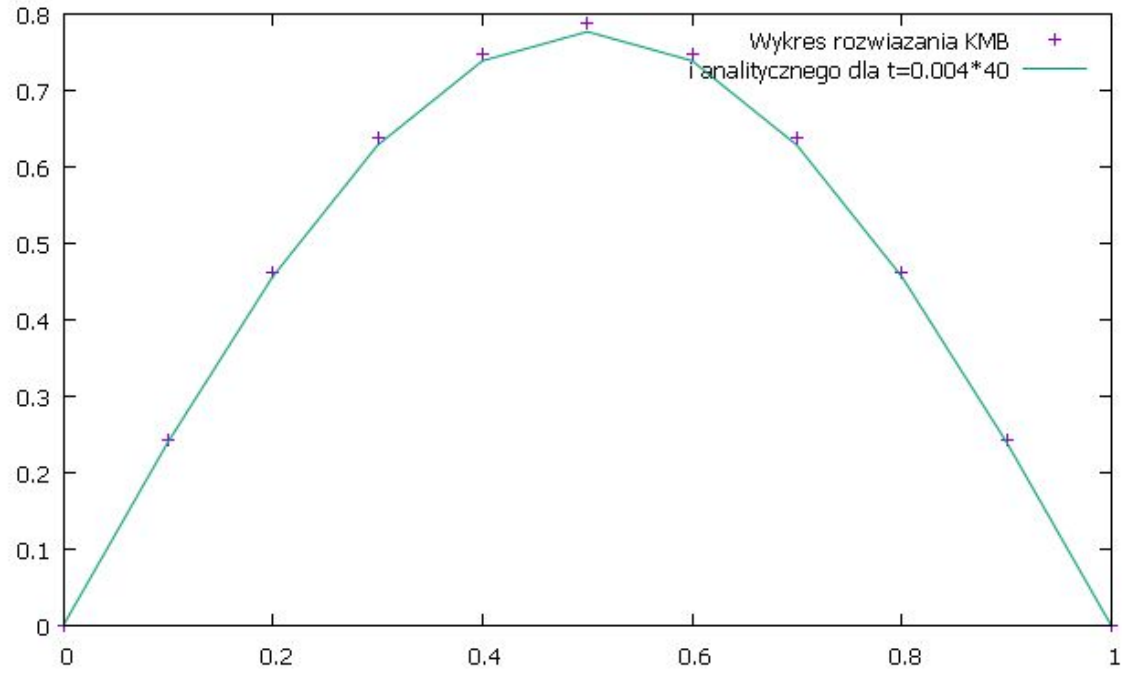
U(x,t)



x

dla **t = 0.16**

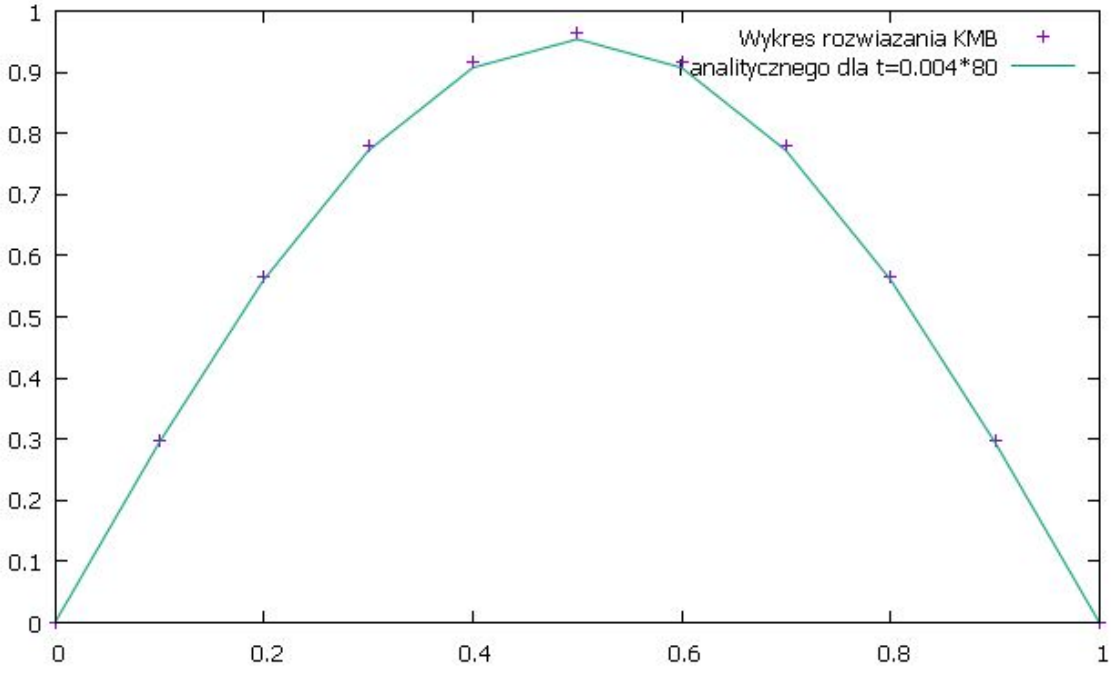
U(x,t)



x

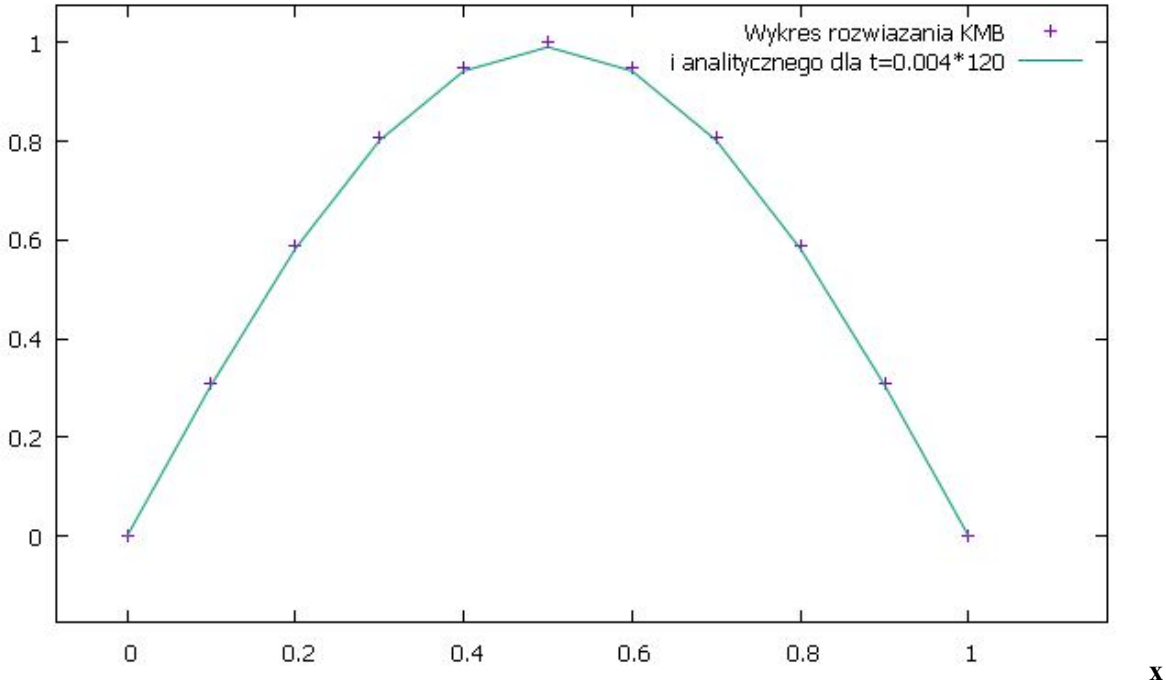
dla **t = 0.32**

U(x, t)



dla **t = 0.48**

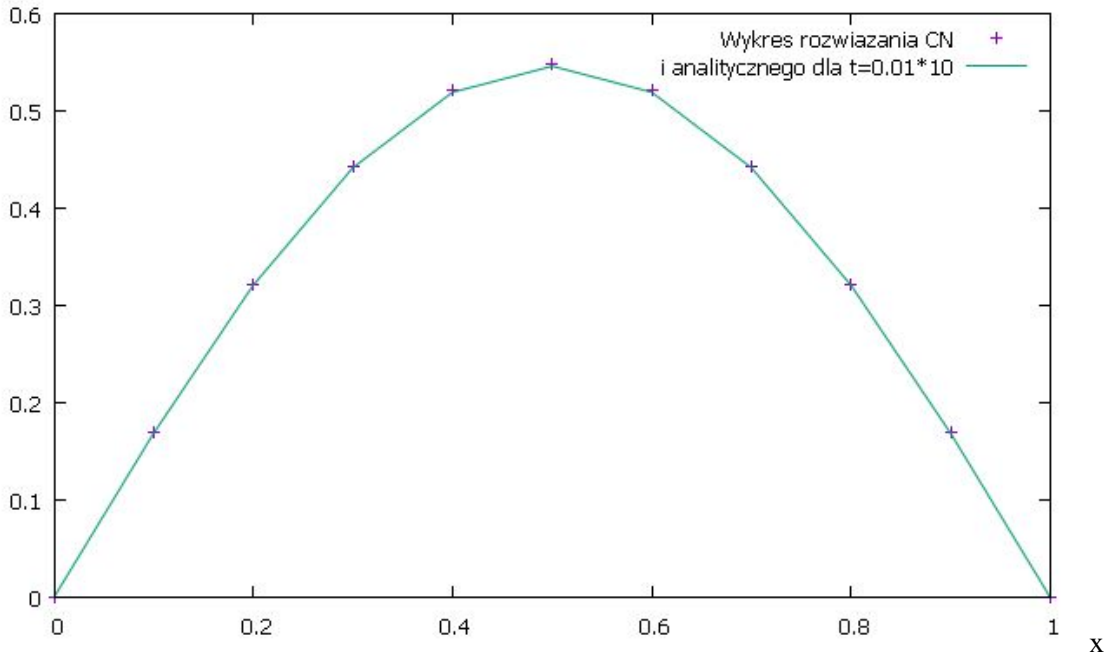
U(x, t)



b)Wykresy rozwiązań numerycznych i analitycznych dla kilku wybranych wartości czasu t z całego przedziału t. Pośrednia metoda Cranka-Nicolson..

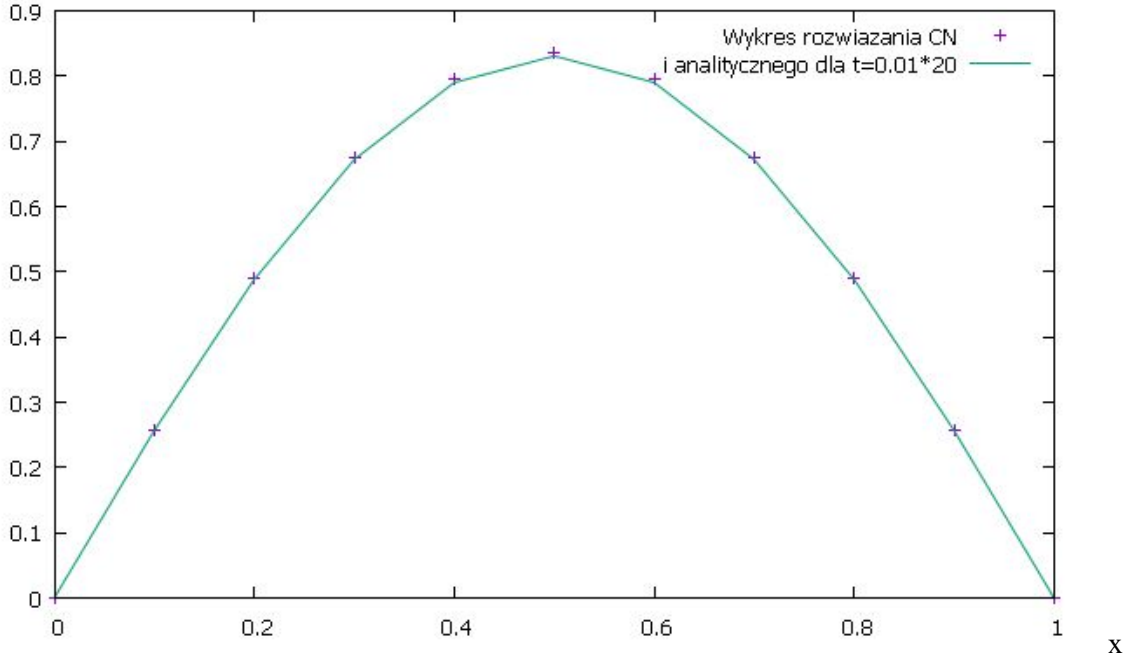
dla **t = 0.1**

U(x,t)



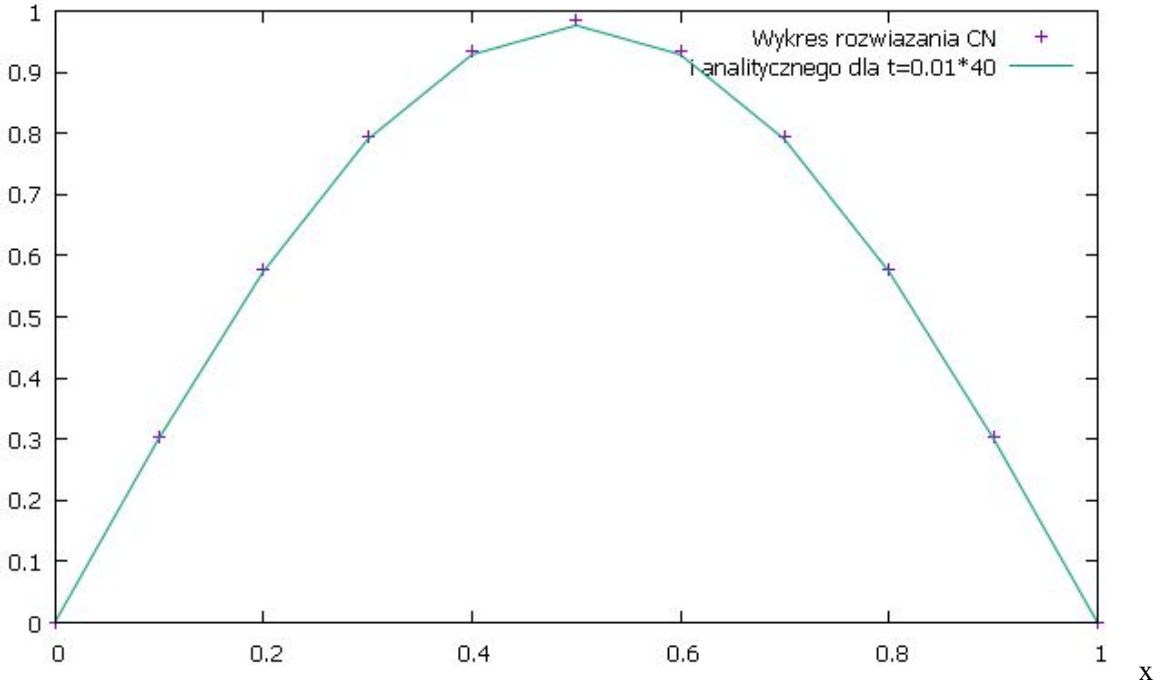
dla **t = 0.2**

U(x,t)



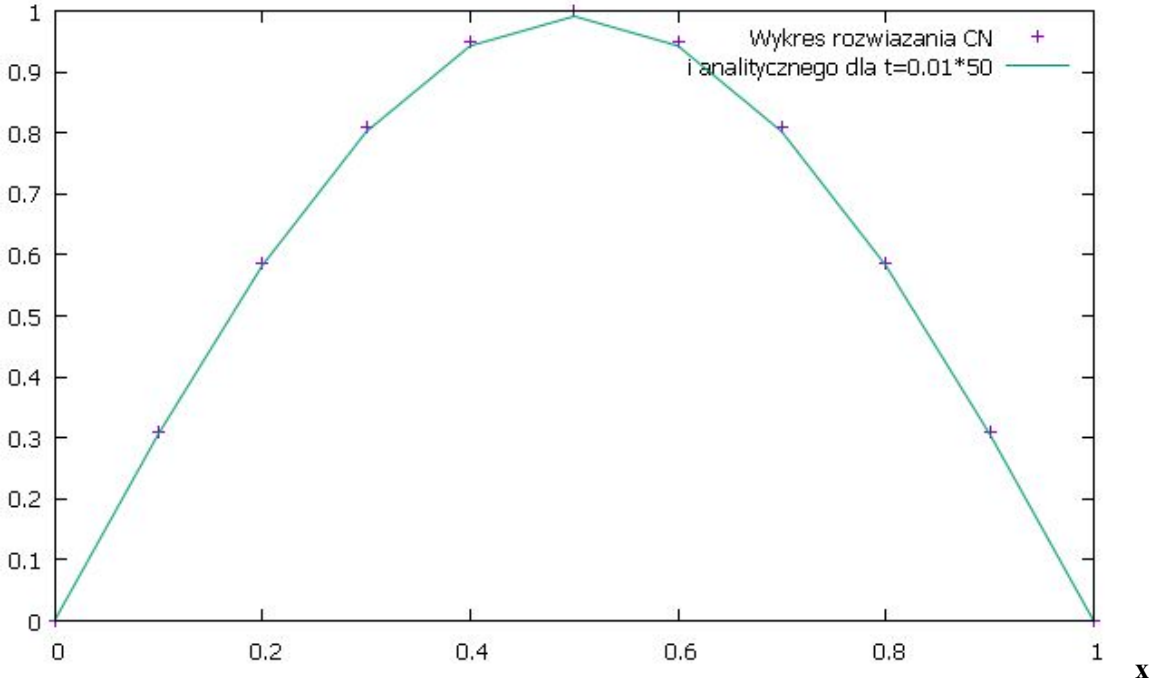
dla **t = 0.4**

U(x, t)



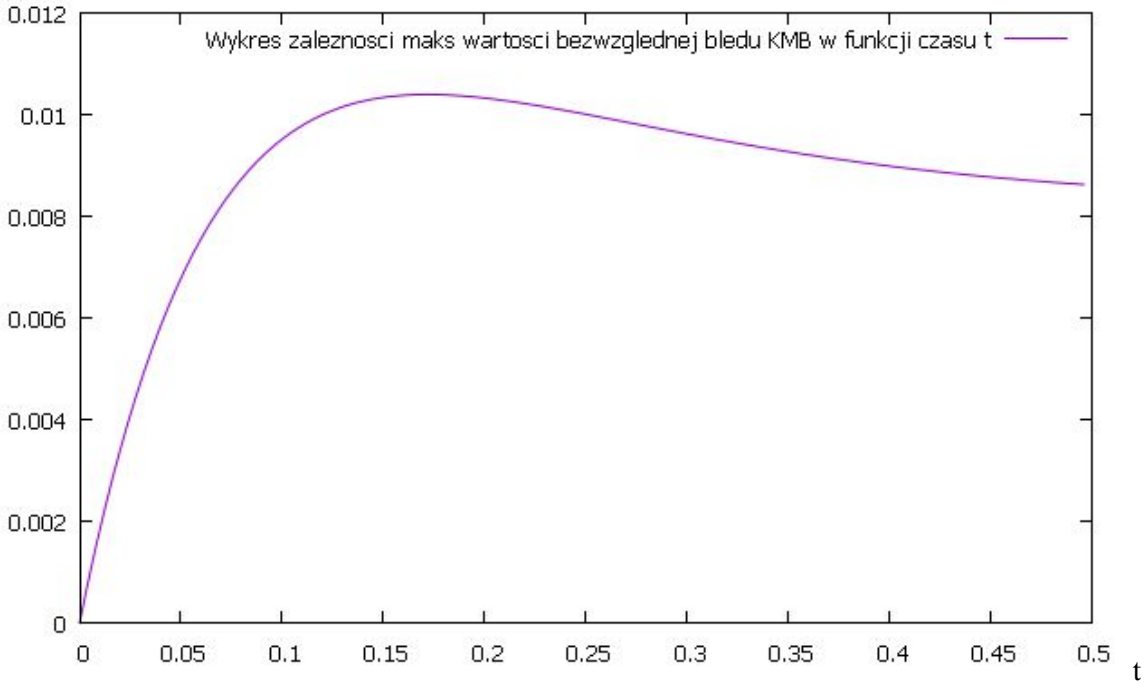
dla **t = 0.5**

U(x, t)



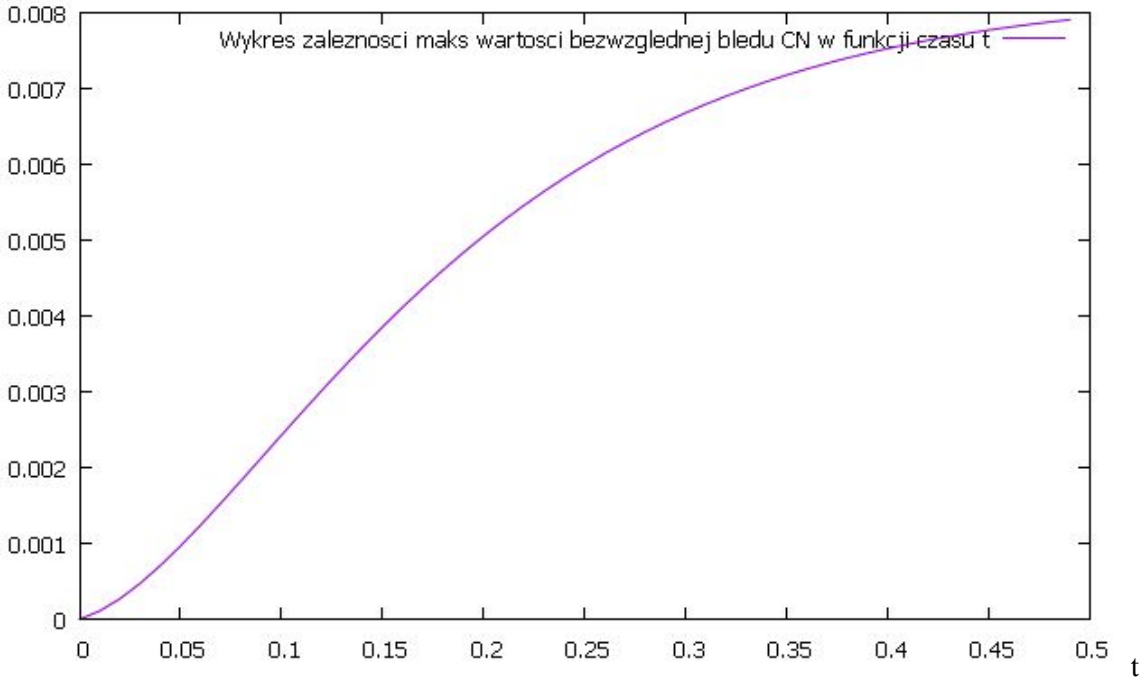
c)Wykres zależności maksymalnej wartości bezwzględnej błędu w funkcji czasu t dla klasycznej metody bezpośredniej.

y - maks wartość bezwzględna błędu



c)Wykres zależności maksymalnej wartości bezwzględnej błędu w funkcji czasu t dla metody pośredniej Cranka-Nicolson.

y - maks wartość bezwzględna błędu



3.Wnioski

Obliczenia numeryczne wartości równania różniczkowego wykorzystujące dyskretyzację klasyczną metodę bezpośrednią lub metodę pośrednią Cranka-Nicolson i dekompozycję LU do rozwiązywania układu równań są dokładne. Nie oznacza to jednak, że nie są obarczone błędem. Dla powyższych metod z krokiem h = 0.1 dla t w przedziale [0, 0.5] błąd nie przekracza 0.008 dla metody CN oraz 0.11 dla metody KMB. Z uwagi na niewielki czas obliczeń jest to zadowalająca dokładność.

Niedokładności w obliczeniach pojawiają się ze względu na błędy maszynowe, błędy obcięcia oraz błędy dyskretyzacji.

Na podstawie wykresu zależności maksymalnej wartości bezwzględnej błędu obserwowanej dla tmax w funkcji kroku przestrzennego h można zauważyć, że im mniejszy krok tym mniejsza wartość błędu maksymalnego a co za tym idzie większa dokładność wyniku.

Ponieważ do dyskretyzacji w zadaniu trzeba było użyć klasycznej metody bezpośredniej, nie potrzebne okazało się użycie algorytmu dekompozycji LU do rozwiązania układu równań. W wypadku użycia metody pośredniej użycie algorytmu LU było wymagane.

Obliczony rząd dokładności dla obu metod pokrywa się z rzędem dokładności teoretycznym.

Program:

```
#include <iostream>
#include <fstream>
#include <math.h>
#include <vector>

class DyskretyzacjaRozwiazanie
{
private:
   double xmin, xmax, tmin, tmax;
   double D, h, lambda, dt;
   int wiersz, kolumna;
   double PI = 3.14159265359;
public:
   DyskretyzacjaRozwiazanie(double lambda) :
      lambda(lambda), xmin(0), xmax(1.0),
      tmin(0), tmax(0.5), D(1.0), h(0.1)
   {
      dt = (lambda*h*h)/D;
      wiersz = static_cast<int>((tmax - tmin)/dt + 1);
      kolumna= static_cast<int>((xmax - xmin)/h + 1);
   }
```

```
std::vector<std::vector<double>> Analityczna(std::vector<std::vector<double>> U)
{
   for(int i = 1; i<U.size(); i++)
   {
      for(int j=1; j<U[0].size()-1; j++)
      {
         U[i][j] = (1 - exp(-(PI*PI)*this->D*dt*i))*sin(PI*j*h);
      }
   }
   return U;

}

std::vector<std::vector<double>>
KlasycznaMetodaBezposrednia(std::vector<std::vector<double>> U){
   U=wypelnijMacierzWarunkiemBrzegowym(U);
   U=wypelnijMacierzWarunkiemPoczatkowym(U);

   for(int i = 1; i < U.size(); i++){
      for(int j = 1; j < U[0].size()-1; j++){

U[i][j]=lambda*U[i-1][j-1]+(1-2*lambda)*U[i-1][j]+lambda*U[i-1][j+1]+dt*(PI*PI)*sin(PI*j*h);
      }
   }
   return U;
}


int wyborCzesciowy(std::vector<std::vector<double>>& macierz, int j, int *indeksy){
   int w;
   for(int i = j + 1; i < macierz.size(); i++) {
      if(fabs(macierz.at(indeksy[i]).at(j)) < fabs(macierz.at(indeksy[i + 1]).at(j))) {
         w = indeksy[i + 1];
      }
      else{
         w = indeksy[i];
      }
   }
   return w;
}

std::vector<std::vector<double>> dekompozycjaLU(std::vector<std::vector<double>> macierz){

   int rozmiar = macierz.size();
   int* indeksy = new int[rozmiar]; //wektor przechowywuj¹cy kolejnoœæ wierszy
   double element;
```

```
    for(int i = 0; i < rozmiar; i++)
      indeksy[i] = i; //numeracja wierszy macierzy po kolei

    //k oznacza k etap
    for(int k = 1; k < rozmiar; k++) {
      element = macierz.at(indeksy[k - 1]).at(k - 1);

      if(element == 0.0) {
        //bierzemy biezacy (dla nas to bedzie 1 czyli kolejny) wiersz i iterujemy do max rozmiaru
        int w = wyborCzesciowy(macierz, indeksy[k], indeksy);
        indeksy[w] = indeksy[k];
        indeksy[k] = w;
        element = macierz[indeksy[k - 1]][k - 1];
      }
      //rozpoczecie iterowania dla tablicy zaczynajacej sie od przekatnej
      for(int i = k; i < rozmiar; i++) {
        //zaczynamy od 'wiersza nizej' bo i=k
        double mnoznik = macierz[indeksy[i]][k - 1] / element;
        macierz[indeksy[i]][k - 1]= mnoznik;
        for(int j = k; j < rozmiar; j++) {
          macierz[indeksy[i]][j] -= macierz[indeksy[k - 1]][j] * mnoznik;
        }
      }
    }
    return macierz;
  }

  std::vector<double> rozwiaz(std::vector<std::vector<double>> macierz, std::vector<double>
wektor){
    int rozmiar = macierz.size();
    std::vector<double> x(rozmiar, 0);
    std::vector<double> y(rozmiar, 0);
    y[0] = wektor[0]; //dla 0 0 ze wzoru
    for(int i = 1; i < rozmiar; i++) {
      y[i] = wektor[i];
      for(int j = 0; j < i; j++) {
        y[i] = y[i] - macierz[i][j] * y[j];
      }
    }
    x[rozmiar - 1] = y[rozmiar - 1] / macierz[rozmiar - 1][rozmiar - 1];
    for(int i = rozmiar - 1; i >= 0; i--) {
      x[i] = y[i];
      for(int j = i + 1; j < rozmiar; j++) {
        //od przekatnej do konca
        x[i] = x[i] - macierz[i][j] * x[j];
```

```
            }
            //koncowe podzielenie przez wartosc na przekatnej
            x[i] = x[i] / macierz[i][i];


        }
        return x;
    }


    std::vector<std::vector<double>> CrankNicolson(std::vector<std::vector<double>> U){
        U=wypelnijMacierzWarunkiemBrzegowym(U);
        U=wypelnijMacierzWarunkiemPoczatkowym(U);
        std::vector<std::vector<double>> A(kolumna, std::vector<double>(kolumna, 0));
        std::vector<double> B;

        A.at(0).at(0) = 1;

        for(int i = 1; i<kolumna-1; i++){
            A.at(i).at(i-1) = lambda/2.0;
            A.at(i).at(i) = -(1+lambda);
            A.at(i).at(i+1) = lambda/2.0;
        }
        A.at(kolumna-1).at(kolumna-1) = 1;
        std::vector<std::vector<double>> LU = dekompozycjaLU(A);
        for(int i = 1; i<wiersz; i++){
            B.clear();
            B.push_back(0);
            for(int j = 1; j<kolumna-1; j++){

B.push_back(-((lambda/2.0)*U.at(i-1).at(j-1)+(1-lambda)*U.at(i-1).at(j)+(lambda/2.0)*U.at(i-1).at(j+
1))-dt*(PI*PI)*sin(PI*j*h));
            }
            B.push_back(0);
            U.at(i) = rozwiaz(LU, B);
        }
        return U;
    }



    std::vector<std::vector<double>> liczMacierzBledu(std::vector<std::vector<double>> U,
std::vector<std::vector<double>> Uanali)
    {
        std::vector<std::vector<double>> blad(U.size(), std::vector<double>(U[0].size(), 0));
        for(int i = 0; i<U.size(); i++){
```

```
            for(int j = 0; j<U[0].size();j++){
                blad.at(i).at(j) = fabs(U[i][j] - Uanali[i][j]);
            }
        }
        return blad;
    }

    std::vector<double> liczWektorBledowMaksOdT(std::vector<std::vector<double>>
macierzBledow)
    {
        std::vector<double> blad(macierzBledow.size(), 0);
        double max;
        for(int i = 0; i<macierzBledow.size(); i++){
            blad[i] = fabs(macierzBledow[i][0]);
            for(int j = 0; j<macierzBledow[0].size();j++){
                if(fabs(blad[i])<fabs(macierzBledow[i][j]))
                    blad[i]=fabs(macierzBledow[i][j]);
            }
        }
        return blad;
    }

    std::vector<std::vector<double>>
wypelnijMacierzWarunkiemBrzegowym(std::vector<std::vector<double>> macierz){
        for(int i = 0; i<macierz.size();i++){
            macierz[i][0] = 0;
            macierz[i][macierz[0].size()-1] = 0;}
        return macierz;}


    std::vector<std::vector<double>>
wypelnijMacierzWarunkiemPoczatkowym(std::vector<std::vector<double>> macierz){
        for(int i = 0; i<macierz[0].size();i++){
            macierz[0][i] = 0;}
        return macierz;}


    void zapiszRozwiazanieTransponowaneDoPliku(std::fstream& plik, std::vector<std::vector<double
>> U){
        std::vector<std::vector<double >> UT = transponujMacierz(U);
        for(int i=0;i<UT.size();i++){
            plik << this->h*i << " ";
            for(int j = 0;j<UT[0].size();j++){
                plik << UT[i][j] << " ";}
        plik << std::endl;}}
```

```
    void zapiszRozwiazanieDoPliku(std::fstream& plik, std::vector<std::vector<double >> U){
        for(int i=0;i<U.size();i++){
            for(int j = 0;j<U[0].size();j++){
                plik << U[i][j] << " ";}
        plik << std::endl;}}

    void zapiszMaxBladDoPliku(std::fstream& plik, std::vector<double> wektor){
        for(int i = 0; i<wektor.size();i++){
            plik << dt*i << " " << wektor.at(i) << " " <<  log10(dt*i) << " " << log10(wektor.at(i)) <<
std::endl;}}

    void zapiszMaxBladTMaksDoPliku(std::fstream& plik, std::vector<double> wektor){
        double k = 0.01;
        for(int i = 0; i<wektor.size();i++){
            plik << k << " " << wektor.at(i) << " " << log10(k) << " " << log10(wektor.at(i)) << std::endl;
            k+=0.01;}}

    std::vector<std::vector<double >> transponujMacierz(std::vector<std::vector<double >> U){
        std::vector<std::vector<double>> UT(U[0].size(), std::vector<double>(U.size(), 0));
        for(int i = 0; i < U.size(); i++)
            for(int j = 0; j < U[0].size(); j++) UT[j][i] = U[i][j];
        return UT;}


    void rysujMacierz(std::vector<std::vector<double>> macierz){
        for(int i = 0; i<macierz.size();i++){
            for(int j = 0; j<macierz[0].size();j++){
                std::cout << macierz.at(i).at(j);}
            std::cout << std::endl;}}


    void rysujWektor(std::vector<double> wektor){
        for(int i = 0; i<wektor.size();i++){
            std::cout << wektor.at(i);}}

    std::vector<std::vector<double>> dajMacierz(){
        std::vector<std::vector<double>> U(wiersz, std::vector<double>(kolumna, 0));
        return U;}



};


int main()
```

```
{
   DyskretyzacjaRozwiazanie KMB(0.4);
   std::vector<std::vector<double>> U1 = KMB.dajMacierz();
   std::fstream fakmb("analityczneKMB.txt", std::ios::out);
   std::fstream fbkmb("KMB.txt", std::ios::out);
   std::fstream fbladkmb("macierzbleduKMB.txt", std::ios::out);
   std::fstream fmaxbladkmb("wektormaxbleduKMB.txt", std::ios::out);

   std::vector<std::vector<double>> AKMB;
   std::vector<std::vector<double>> BKMB;
   std::vector<std::vector<double>> bladKMB;
   std::vector<double> maksBladKMB;

   AKMB = KMB.Analityczna(U1);
   BKMB = KMB.KlasycznaMetodaBezposrednia(U1);
   bladKMB = KMB.liczMacierzBledu(BKMB,AKMB);
   maksBladKMB = KMB.liczWektorBledowMaksOdT(bladKMB);

   KMB.zapiszRozwiazanieTransponowaneDoPliku(fakmb,AKMB);
   KMB.zapiszRozwiazanieTransponowaneDoPliku(fbkmb,BKMB);
   KMB.zapiszRozwiazanieTransponowaneDoPliku(fbladkmb,bladKMB);
   KMB.zapiszMaxBladDoPliku(fmaxbladkmb, maksBladKMB);

   //

   DyskretyzacjaRozwiazanie CN(1.0);
   std::vector<std::vector<double>> U2 = CN.dajMacierz();

   std::fstream facn("analityczneCN.txt", std::ios::out);
   std::fstream fbcn("CN.txt", std::ios::out);
   std::fstream fbladcn("macierzbleduCN.txt", std::ios::out);
   std::fstream fmaxbladcn("wektormaxbleduCN.txt", std::ios::out);

   std::vector<std::vector<double>> ACN;
   std::vector<std::vector<double>> BCN;
   std::vector<std::vector<double>> bladCN;
   std::vector<double> maksBladCN;

   ACN = CN.Analityczna(U2);
   BCN = CN.CrankNicolson(U2);
   bladCN = CN.liczMacierzBledu(BCN,ACN);
   maksBladCN = CN.liczWektorBledowMaksOdT(bladCN);

   CN.zapiszRozwiazanieTransponowaneDoPliku(facn,ACN);
   CN.zapiszRozwiazanieTransponowaneDoPliku(fbcn,BCN);
   CN.zapiszRozwiazanieTransponowaneDoPliku(fbladcn,bladCN);
```

```
    CN.zapiszMaxBladDoPliku(fmaxbladcn, maksBladCN);

}
```