|  |
| --- |
| ***Metody Obliczeniowe*** |
| ***Projekt D*** |
|  |

|  |
| --- |
|  |

1. ***Wstęp:***
   1. **Temat**: Rozwiązywanie równania różniczkowego cząstkowego metodami numerycznymi.
   2. **Równanie różniczkowe cząstkowe:**

Określone dla współrzędnej przestrzennej oraz czasu .

* Warunek początkowy:
* Warunki brzegowe: ( = 1

Zagadnienie to może opisywać transport ciepła w ośrodku wokół kuli o promieniu r, przy współczynniku transportu ciepła D, po raptownym obniżeniu temperatury w chwili t = 0.

* Rozwiązanie analityczne:

Gdzie: , a jest tzw. funkcją błędu:

* 1. **Założenia:**

Do obliczeń numerycznych:

* Przedział nieskończony ***x***należy zastąpić przedziałem skończonym
* Drugi warunek brzegowy zastąpić warunkiem:

Do obliczenia funkcji z dokładnością bliską dokładności maszynowej dla zmiennych

typu **double** należy zastosować pakiet **CALERF** udostępniony przez prowadzącego zajęcia.

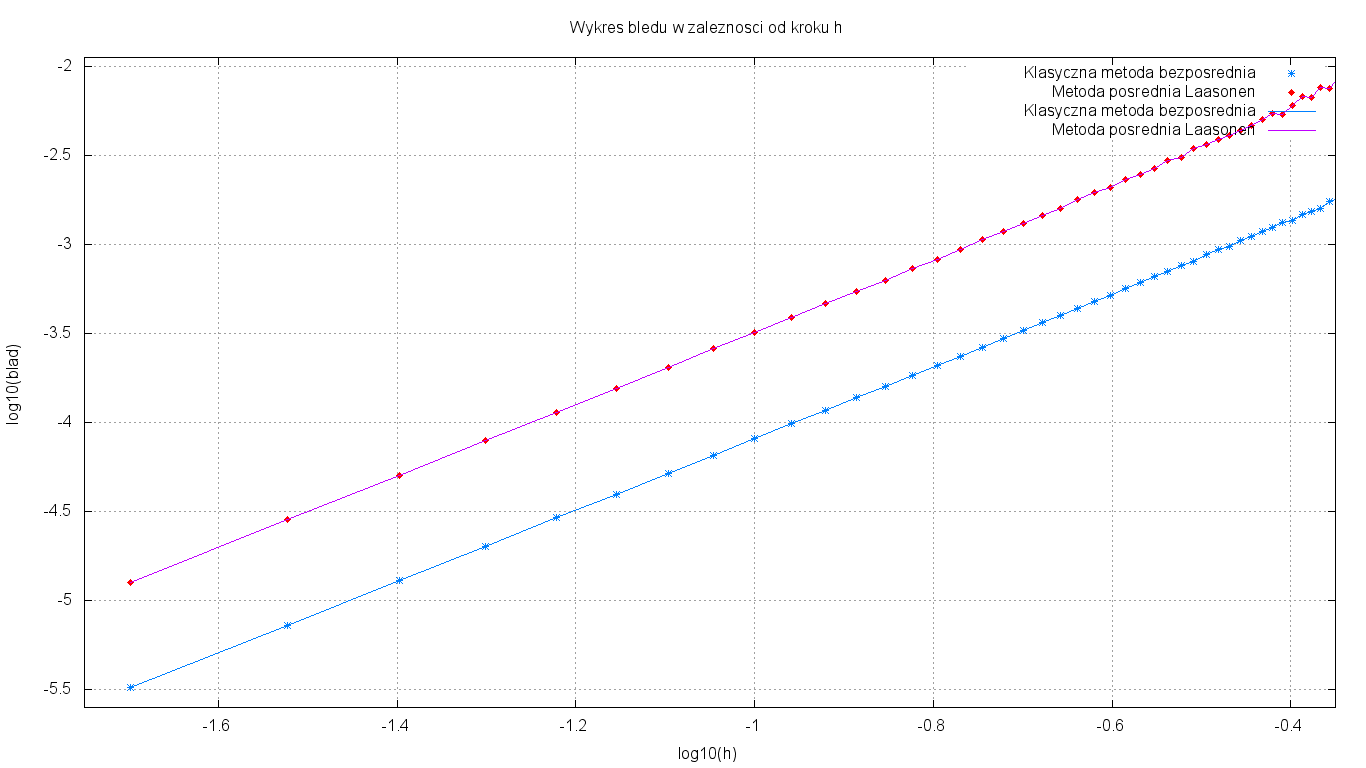
* 1. **Parametry:**
  2. **Algorytmy:**
* Dyskretyzacja:
  + Klasyczna Metoda Bezpośrednia
  + Metoda p\Pośrednia Laasonen
* Rozwiązania algebraiczne układów równań licznikowych:
  + Algorytm Thomasa

1. ***Dyskretyzacja***:

* *Dla Metody Pośredniej Laasonen:*

* *Dla Klasycznej Metody Bezpośredniej:*

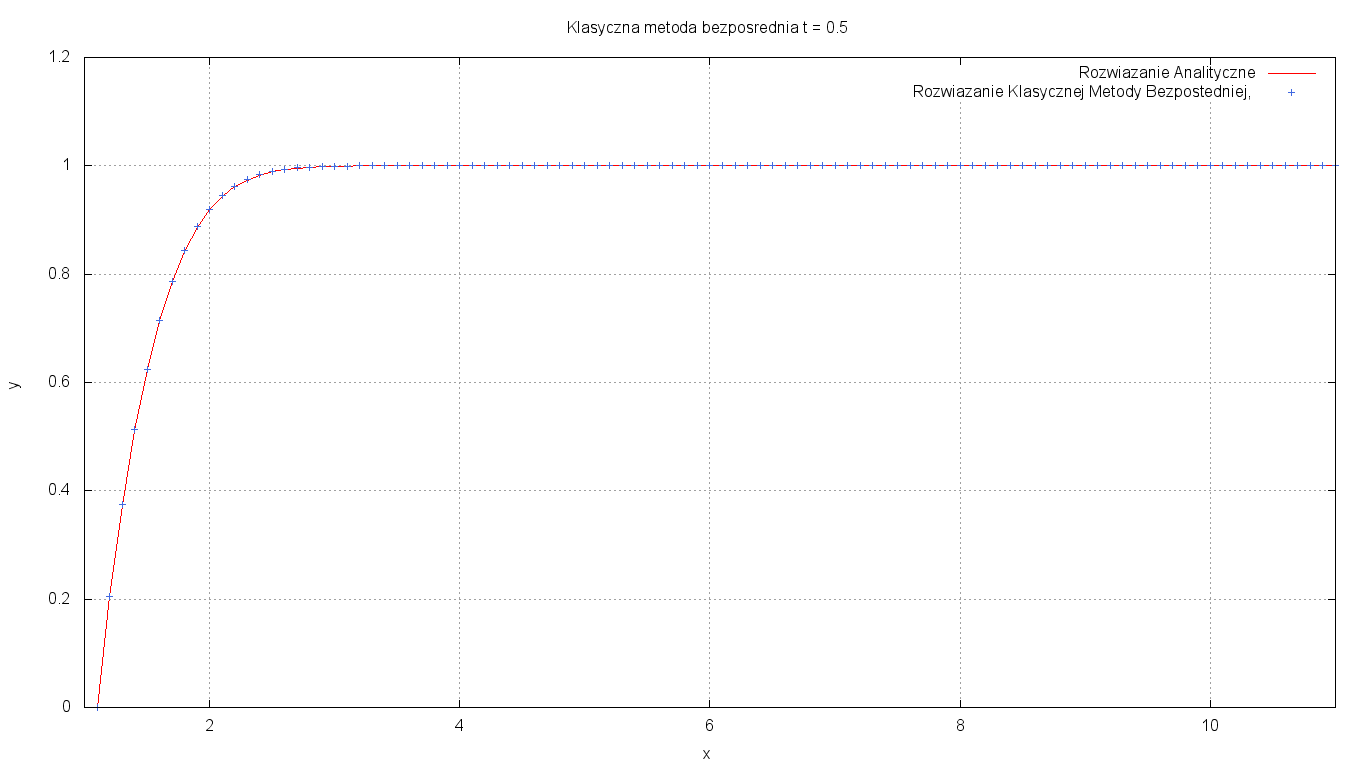
1. ***Wykresy***:
   1. *Wykres maksymalnej wartości bezwzględnej błędu obserwowanej dla* , w funkcji kroku przestrzennego h:

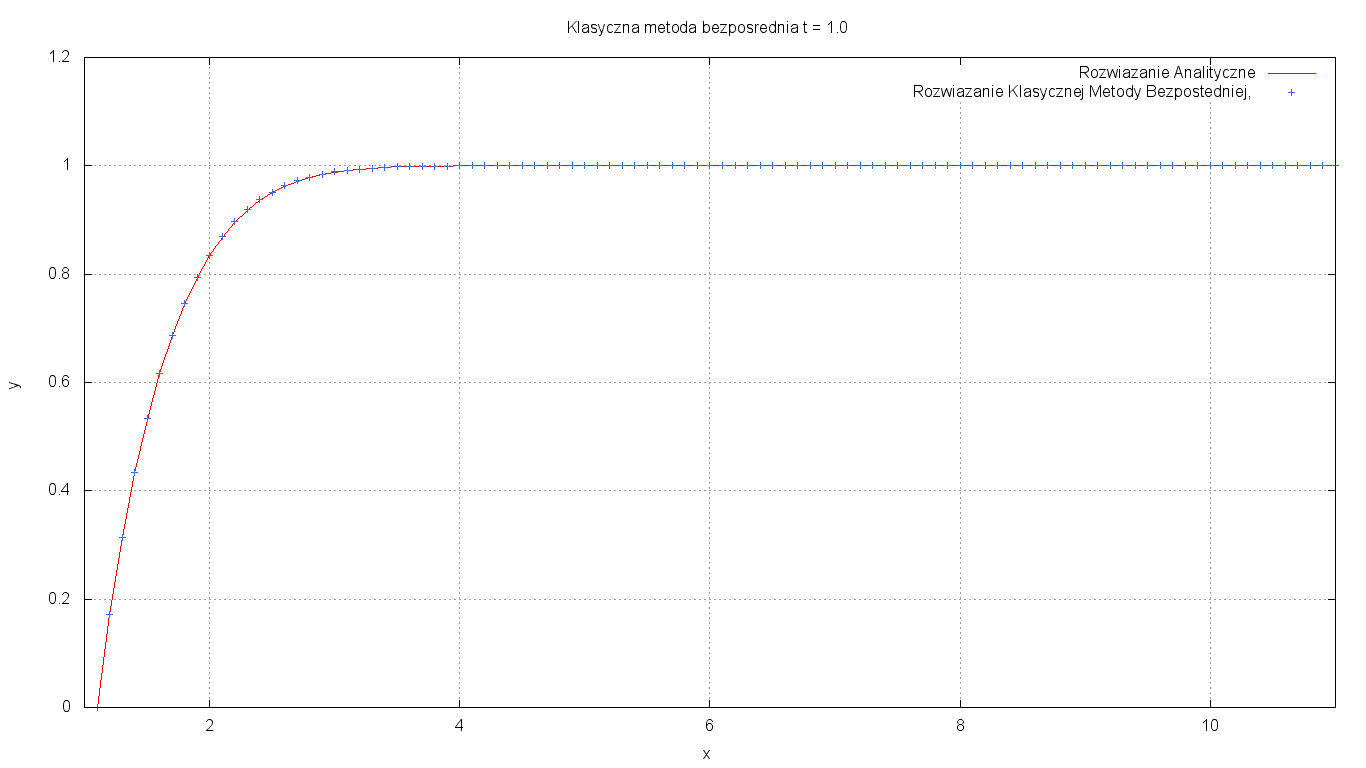


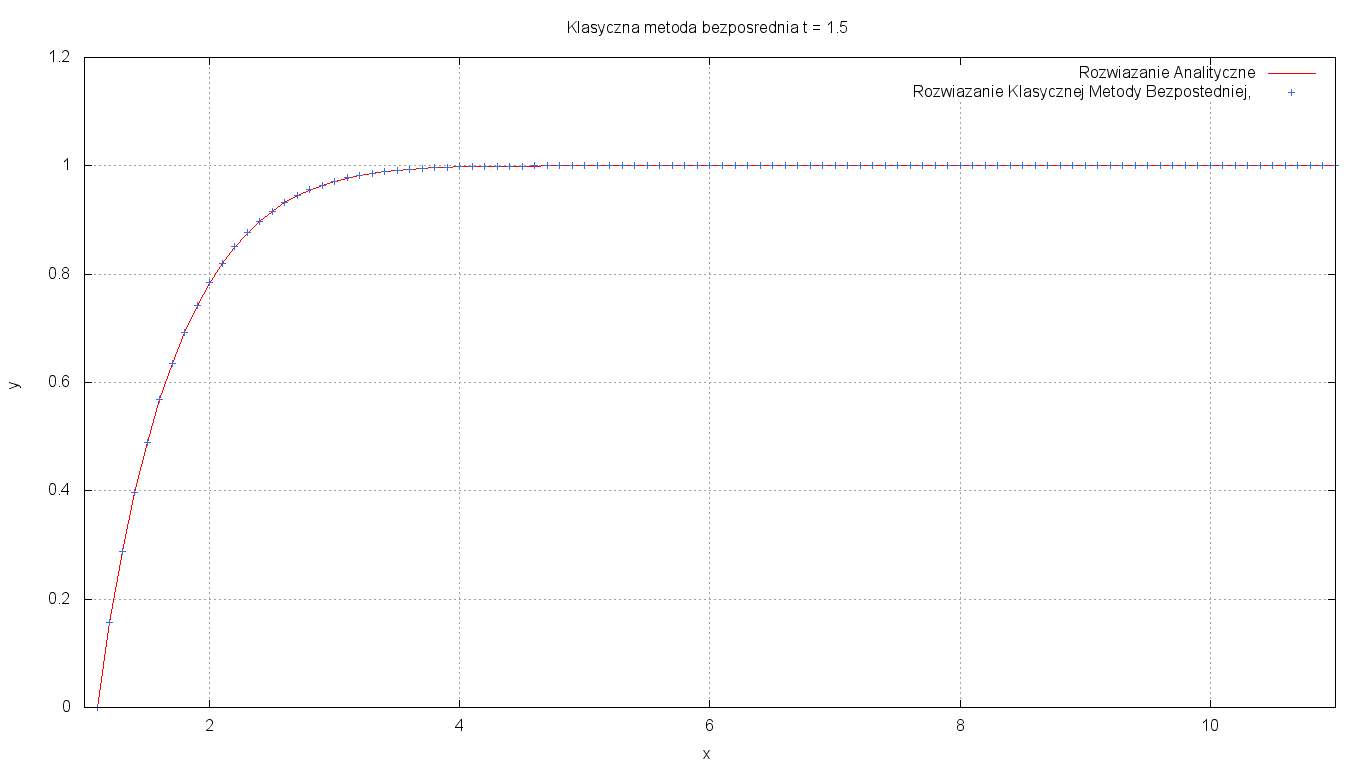
Wykres przedstawia zależność błędu od kroku (zakres od 0.5 do 0.01). Na jego podstawie można zauważyć, że rzędy dokładności potwierdzają przypuszczenia teoretyczne.

|  |  |  |
| --- | --- | --- |
|  | **Dane (dla h = 0.1)** | **Teoretyczny  rząd dokładności** |
| *Klasyczna Metoda Bezpośrednia* |  |  |
| *Metoda pośrednia Laasonen* |  |  |

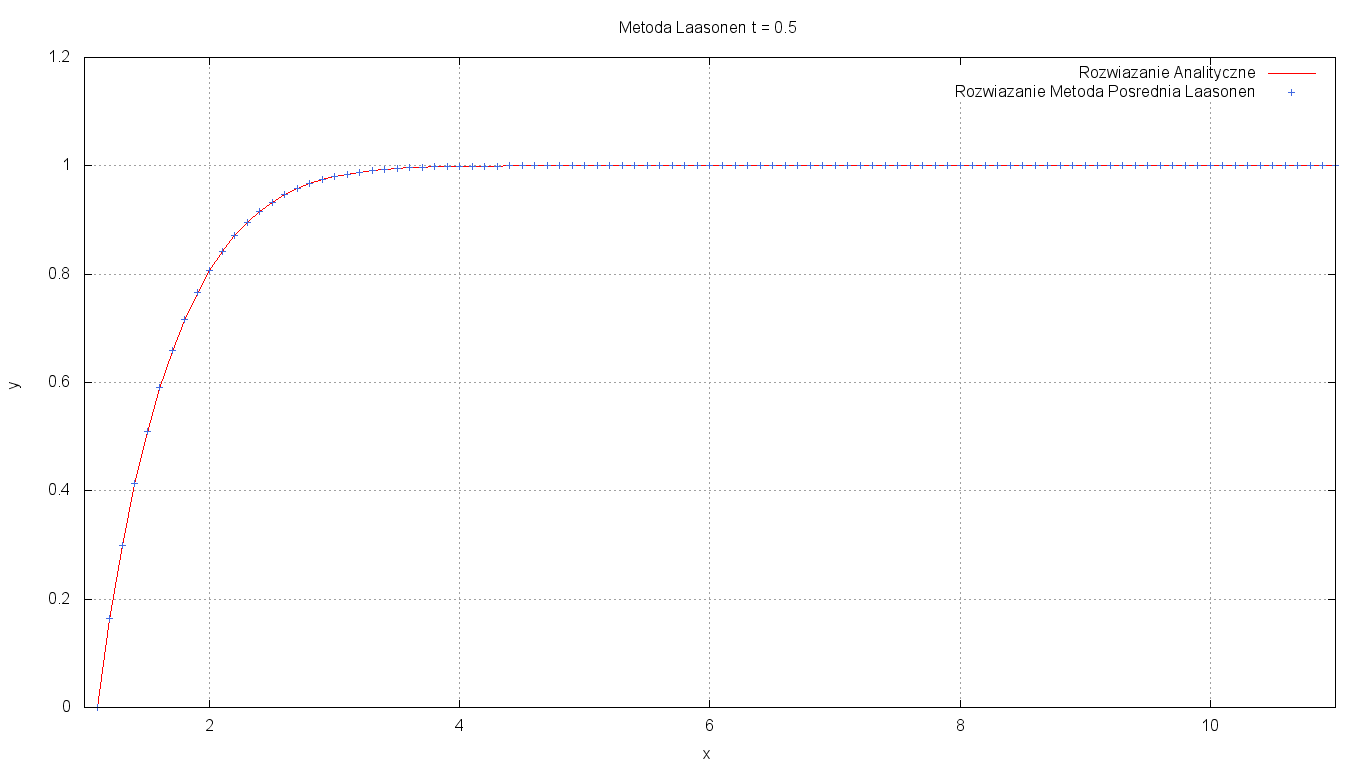
* 1. Wykres rozwiązań numerycznych i analitycznych dla kilku wybranych wartości czasu t:
     + Klasyczna Metoda Bezpośrednia:

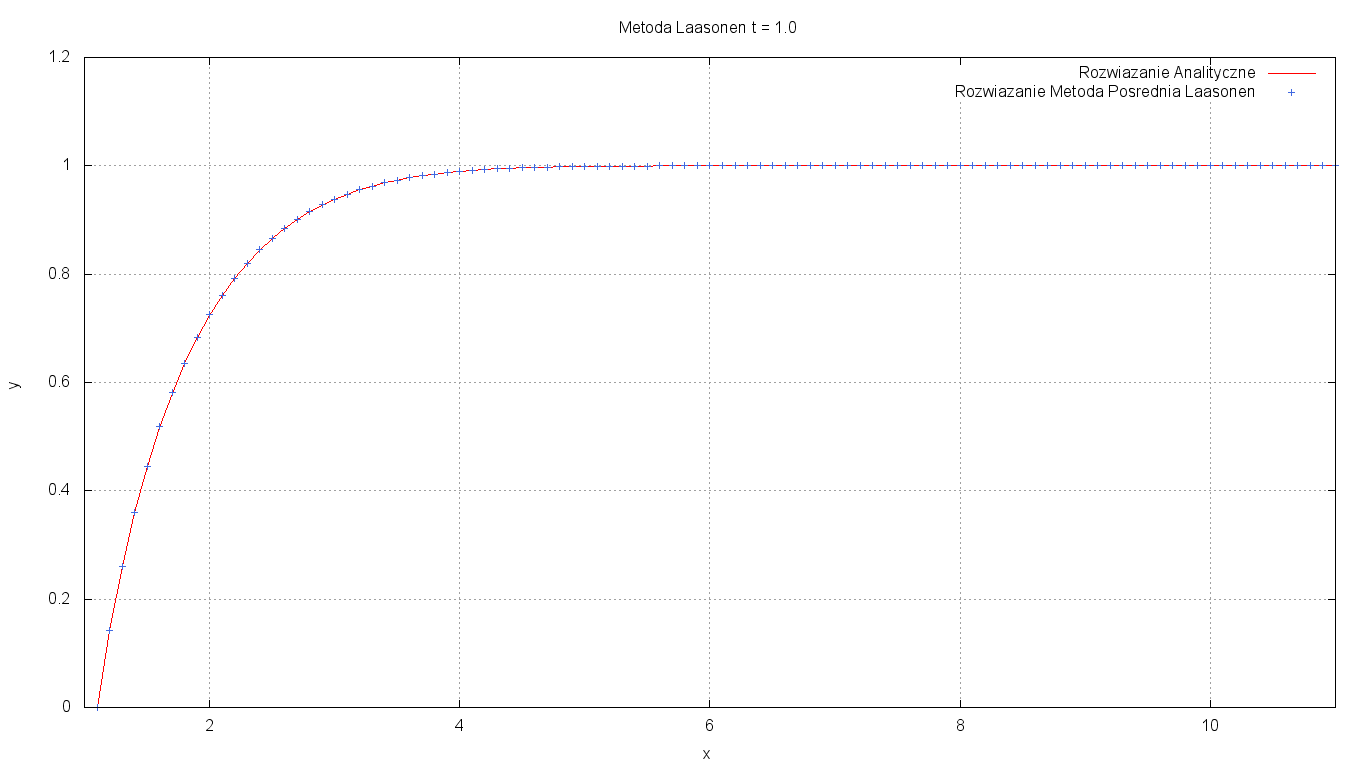


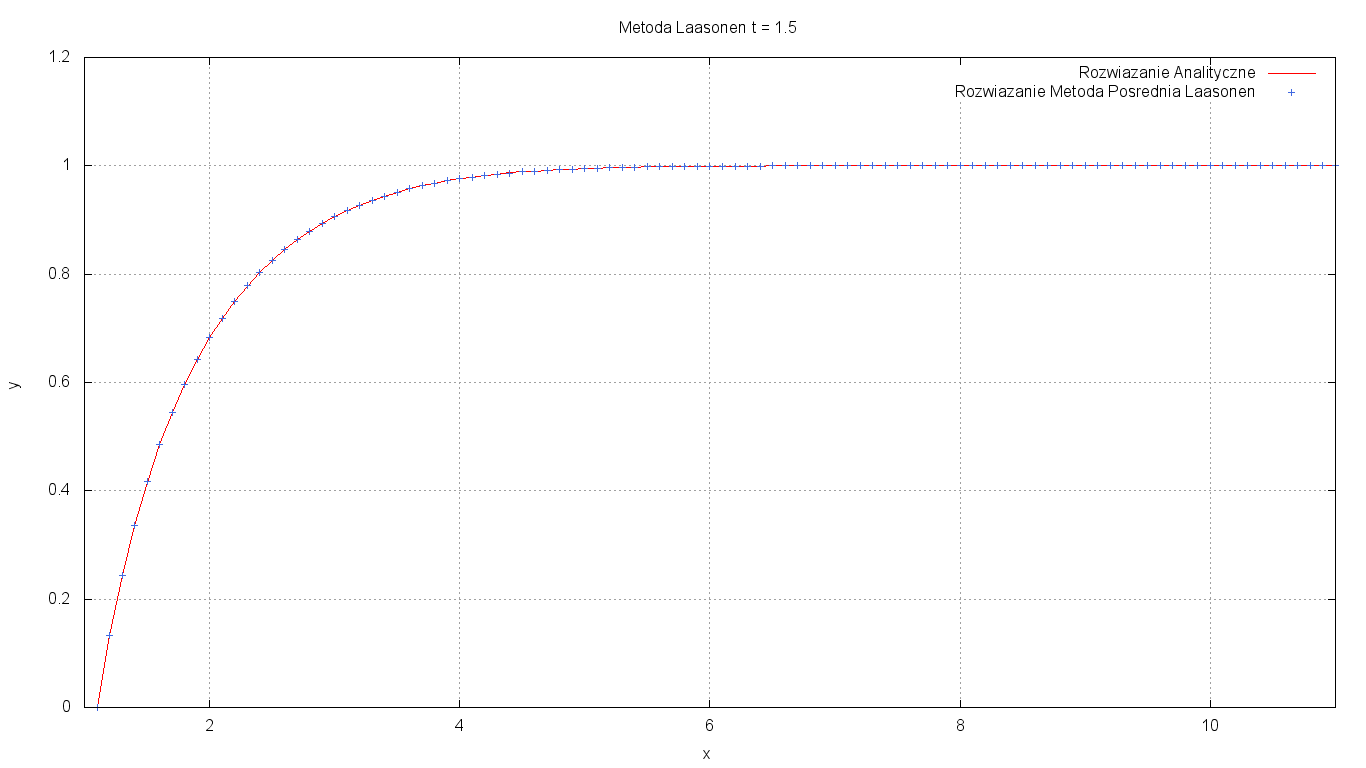




* + - Metoda pośrednia Laasonen:

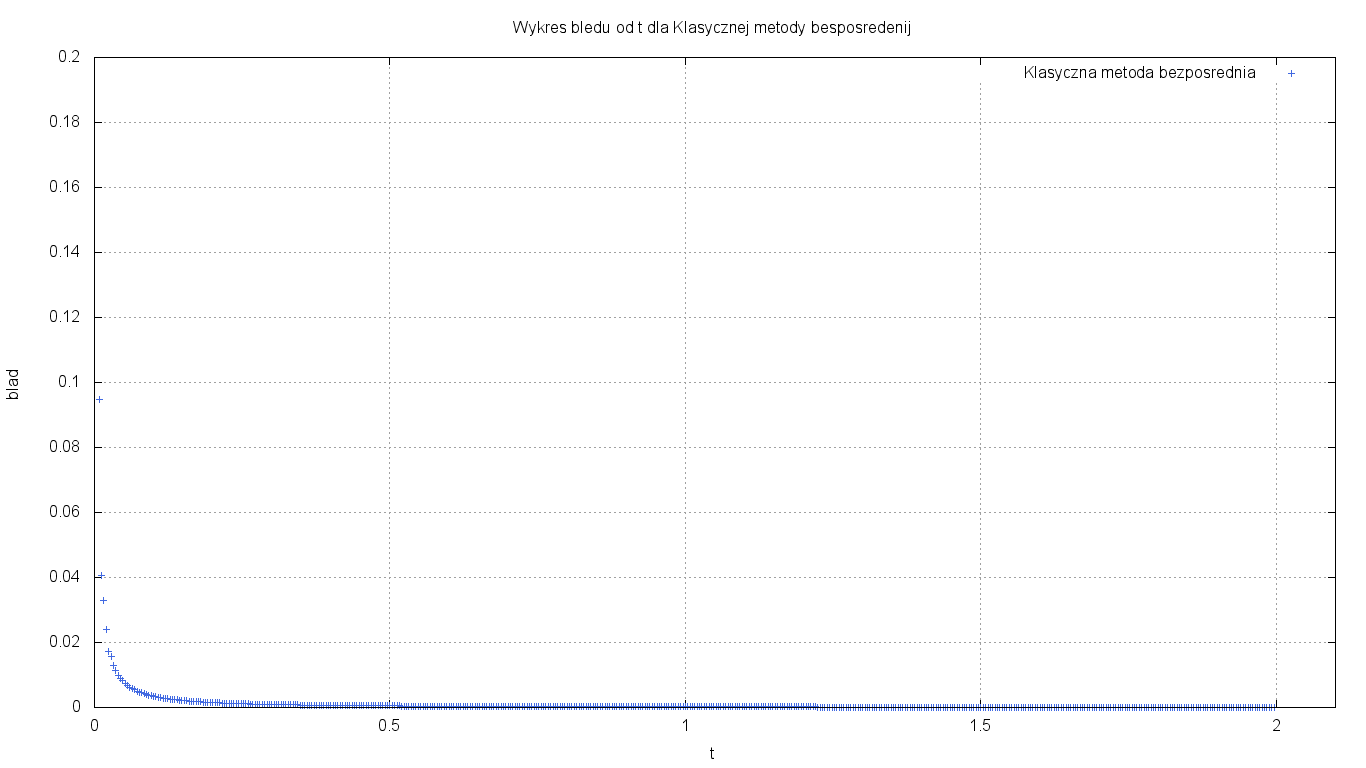




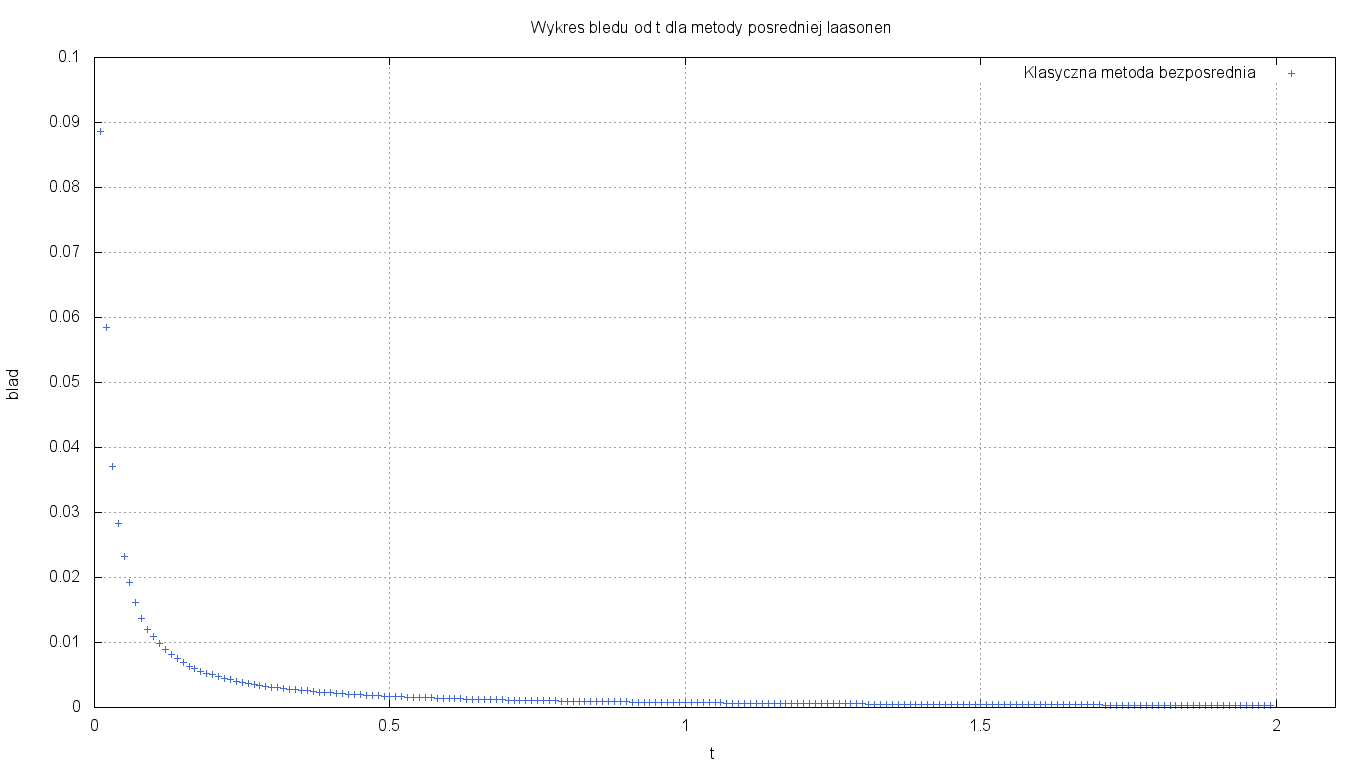


Dla wszystkich wykresów, rozwiązania analityczne pokrywają się z rozwiązaniami numerycznymi.

* 1. Wykres zależności maksymalnej wartości bezwzględnej błędu w funkcji czasu t:
     + Klasyczna Metoda Bezpośrednia:



* + - Metoda Pośrednia Laasonen:



Wartość błędu wyraźnie spada raz z obliczeniami kolejnych poziomów t. Najbardziej zauważalny jest ona dla małych wartości t, następnie błąd maleje liniowo. W przypadku gdy błąd rósłby wraz ze wzrostem t, świadczyło by o błędnym wyprowadzaniu i wykorzystaniu wzorów. Metoda Laasonen wykazuje nieco większa zbieżności dla początkowych poziomów czasowych.

1. ***Wnioski Końcowe:***

Obliczania numeryczne wartości równania różniczkowego wykorzystujące dyskretyzację

Klasyczną metodę bezpośrednią i Metodę pośrednią Laasonen oraz Algorytm Thomasa do

rozwiązania układu równań są dokładne. Nie oznacza to jednak, ze nie są one obarczone

błędami. Najważniejsza częścią błędu jest błąd maszynowy oraz błąd przybliżeń. Nie jesteśmy

wstanie całkowicie ich wyeliminować. Dla powyższych metod (dla kroku h = 0.1 i t > 1) błąd

jest rzędu co jest zadowalającą dokładnością – przy niewielkim czasie obliczeń.

1. Program:

|  |
| --- |
| #include <iostream>  #include <Windows.h>  #include <math.h>  #include "calerf.h"  #include <iomanip>  #include <fstream>  using namespace std;  //Stałe Wspolczynniki  const double t\_min = 0.0, t\_max = 2.0;  const double r = 1.0, a = 10.0;  const double D = 1.0, LAMBDA\_KMB = 0.4, LAMBDA\_LAASONEN = 1.0;  // t\_size il poziomow czasowych dla laasonen i kmb  int t\_size\_laasonen, t\_size\_kmb, x\_size;  //pliki do zapisania danych do wykresu  fstream dane\_wykres\_analityczny, dane\_blad\_tmax, dane\_blad\_t\_laasonen, dane\_blad\_kmb;  //Funkcja obliczajaca wartosci analityczne dla Klasycznej metody bezposredeniej  void rozwiazanie\_analityczne\_K( double \*\*A, double dt , double h){  double tt = dt, dh;  for ( int i = 1; i < t\_size\_kmb; i++ ){  dh = 1;  for ( int j = 0; j < x\_size; j++ ){  A[i][j] = 1.0 -(r / dh) \* calerf\_erfc((dh - r) / (2.0 \* sqrt(D \* tt)));  dh += h;  }  tt += dt;  }  }  //Funkcja obliczajaca wartosci analityczne dla Metody posredniej laasonen  void rozwiazanie\_analityczne\_L( double \*\*A, double dt, double h ){  double tt = dt, dh;  for ( int i = 1; i < t\_size\_laasonen; i++ ){  dh = 1;  for ( int j = 0; j < x\_size; j++ ){  A[i][j] = 1.0 - (r /dh) \* calerf\_erfc((dh - r) / (2.0 \* sqrt(D \* tt ) ));  dh += h;  }  tt += dt;  }}  //Algorytm Thomasa  void thomas\_redukcja(double \*l, double \*d, double \*u){  for ( int i = 0; i < x\_size; i++ ){  d[i + 1] = d[i + 1] - l[i] \* ( u[i] / d[i] );  l[i] = l[i] / d[i];  }  }  void thomas\_rozwiazanie\_ukladu( double \*l, double \*d, double \*u, double \*b, double \*x ){  for ( int i = 1; i < x\_size ; i++ )  b[i] = b[i] - l[i - 1] \* b[i - 1];  x[x\_size - 1] = b[x\_size - 1] / d[x\_size - 1];  for ( int i = x\_size - 2; i >= 0; i-- )  x[i] = ( b[i] - u[i] \* x[i + 1] ) / d[i];  }  //obliczenie odpowienich wartosc do macierzy trojdiagonalnej - dyskretyzacja  void dyskretyzacja\_Laasonen( double \*l, double \*d, double \*u, double h ){  double dh = r + h;  d[0] = 1.0;  u[0] = 0.0;  for ( int i = 1; i <= x\_size - 2; i++ ){  l[i - 1] = LAMBDA\_LAASONEN \* ( 1.0 - ( h / dh ) ); //wektor l  d[i] = -( 1.0 + ( 2 \* LAMBDA\_LAASONEN ) ); //wektor d  u[i] = LAMBDA\_LAASONEN \* ( 1.0 + h / dh ); //wektor u  dh += h;  }  d[x\_size - 1] = 1.0;  l[x\_size - 2] = 0.0;  }  // Funkcja realizujaca Metoda bezposrednia Laasonen  void metoda\_posrednia\_laasonen( double \*\* A, double h ){  double \*l, \*d, \*u, \*b, \*x;  l = new double[x\_size - 1];  d = new double[x\_size];  u = new double[x\_size - 1];  b = new double[x\_size];  x = new double[x\_size];  dyskretyzacja\_Laasonen(l,d,u, h);  thomas\_redukcja( l, d, u );  for ( int i = 1; i < t\_size\_laasonen; i++ ){  for ( int j = 1; j < x\_size - 1; j++ )  b[j] = -A[i - 1][j]; // wektor b  b[0] = A[i][0];  b[x\_size - 1] = A[i][x\_size - 1];  thomas\_rozwiazanie\_ukladu( l, d, u, b, x );  for ( int k = 1; k < x\_size - 1 ; k++ )  A[i][k] = x[k]; //wypelnianie macierzy aktualnym poziomem czasowym  }  delete[] l;  delete[] d;  delete[] u;  delete[] x;  delete[] b;  }  // Funkcja realizujaca Klasyczna metode bezposrednia  void metoda\_KMB( double \*\*A, double h ){  double dh;  for ( int i = 1; i < t\_size\_kmb; i++ ){  dh = r + h;  for ( int j = 1; j < x\_size - 1; j++ ){  //obliczanie nowego poziomu czasowego zgodnie z wzorem  A[i][j] = A[i - 1][j - 1] \* LAMBDA\_KMB \* ( 1.0 - h / dh )  + A[i - 1][j] \* ( 1.0 - ( 2.0 \* LAMBDA\_KMB ) )  + A[i - 1][j + 1] \* LAMBDA\_KMB \* ( 1.0 + h / dh );  dh += h;  }  }  }  //warynek brzegowy  double prawy( double t ){  return 1.0 - ( r / ( r + a ) ) \* calerf\_erfc( a / ( 2.0 \* sqrt( D \* t ) ) );  }  //Funkcja realizujaca zadanie  void rozwiazanie()  {  dane\_blad\_tmax.open( "Dane\_blad\_tmax.txt", fstream::out );  double blad\_laasonen, blad\_kmb, max\_blad\_lassonen, max\_blad\_kmb, h = 0.5;    for ( h; h > 0.01; h -= 0.01 ){  double dt\_laasonen = ( LAMBDA\_LAASONEN \* h \* h ) / D;  x\_size = a / h;  //Metoda\_lasonen\_dane  double \*\*Analityczne\_L, \*\*Laasonen;  t\_size\_laasonen = t\_max / dt\_laasonen ;  Analityczne\_L = new double \*[t\_size\_laasonen];  Laasonen = new double \*[t\_size\_laasonen];  for ( int i = 0; i < t\_size\_laasonen; i++ ){  Analityczne\_L[i] = new double[x\_size];  Laasonen[i] = new double[x\_size];  }  //Uzupelnianie tablic warunkami poczatkowymi  //Warunek poczatkowy  for ( int i = 0; i < x\_size; i++ )  Laasonen[0][i] = 1.0;    //1 Warunek brzegowy  for ( int i = 1; i < t\_size\_laasonen; i++ )  Laasonen[i][0] = 0.0;    //2 Warunek brzegowy  double tt = dt\_laasonen;  for ( int i = 1; i < t\_size\_laasonen; i++ ){  Laasonen[i][x\_size - 1] = prawy( tt );  tt += dt\_laasonen;  }  //Metoda\_KMB  double \*\*Analityczne\_K, \*\*KMB;  double dt\_kmb = ( LAMBDA\_KMB \* h \*h ) / D;  t\_size\_kmb = t\_max / dt\_kmb ;  Analityczne\_K = new double \*[t\_size\_kmb];  KMB = new double \*[t\_size\_kmb];  for ( int i = 0; i < t\_size\_kmb; i++ ){  Analityczne\_K[i] = new double[x\_size];  KMB[i] = new double[x\_size];  }  //Uzupelnianie tablic warunkami poczatkowymi  //Warunek poczatkowy  for ( int i = 0; i < x\_size; i++ )  KMB[0][i] = 1.0;    //1 Warunek brzegowy  for ( int i = 1; i < t\_size\_kmb; i++ )  KMB[i][0] = 0.0;    //2 Warunek brzegowy  tt = dt\_kmb;  for ( int i = 1; i < t\_size\_kmb; i++ ){  KMB[i][x\_size - 1] = prawy( tt );  tt += dt\_kmb;  }  //wpelnienie tablic poziomami czasowymi  rozwiazanie\_analityczne\_L( Analityczne\_L, dt\_laasonen, h);  metoda\_posrednia\_laasonen( Laasonen, h );  rozwiazanie\_analityczne\_K( Analityczne\_K, dt\_kmb, h );  metoda\_KMB( KMB, h );  //gdy h == 0.1 zapisuje obliczenia do wykresu funkcji  // oraz licze maksymalna wartosc bezwgledna bledu w funkcji czasu t  if ( h < 0.101 && h > 0.099 )  {  int index = 150;  double dh = r + h;  dane\_wykres\_analityczny.open( "dane\_wykres\_analityczny.txt",  fstream::out );  for ( int i = 0; i < x\_size; i++ ){  dane\_wykres\_analityczny << dh << " " << Analityczne\_L[index][i]  << " " << Laasonen[index][i] <<" "<< Analityczne\_K[index][i]  <<" "<< KMB[index][i] << endl;    dh += h;  }  dane\_wykres\_analityczny.close();  double blad\_laasonen, blad\_kmb, max\_blad\_lassonen, max\_blad\_kmb;  dane\_blad\_t\_laasonen.open( "dane\_blad\_t\_laasonen.txt", fstream::out );  for ( int i = 1; i < t\_size\_laasonen; i++ ){  max\_blad\_lassonen = 0.0;  for ( int j = 0; j < x\_size; j++ ){  blad\_laasonen = fabs(Analityczne\_L[i ][j] - Laasonen[i][j]);  if ( blad\_laasonen > max\_blad\_lassonen )  max\_blad\_lassonen = blad\_laasonen;  }  dane\_blad\_t\_laasonen << i \* dt\_laasonen << " " <<  max\_blad\_lassonen << endl;  }  dane\_blad\_t\_laasonen.close();  dane\_blad\_kmb.open( "dane\_blad\_kmb.txt", fstream::out );  for ( int i = 1; i < t\_size\_kmb; i++ ){  max\_blad\_kmb = 0.0;  for ( int j = 0; j < x\_size; j++ ){  blad\_kmb = fabs( Analityczne\_K[i][j] - KMB[i][j] );  if ( blad\_kmb > max\_blad\_kmb )  max\_blad\_kmb = blad\_kmb;  }  dane\_blad\_kmb << i\*dt\_kmb << " " << max\_blad\_kmb << endl;  }  }  max\_blad\_lassonen = 0.0;  max\_blad\_kmb = 0.0;  //Maksymalna wartosc bezwzglednego bledu obserwacji dla t\_max w funkcji kroku  // przestrzennnego h  for ( int i = 0; i < x\_size; i++ ){  blad\_laasonen = fabs( Analityczne\_L[t\_size\_laasonen - 1][i] -  Laasonen[t\_size\_laasonen - 1][i] );  if ( blad\_laasonen > max\_blad\_lassonen )  max\_blad\_lassonen = blad\_laasonen;  blad\_kmb = fabs(Analityczne\_K[t\_size\_kmb-1][i]- KMB[t\_size\_kmb- ][i]);  if ( blad\_kmb > max\_blad\_kmb )  max\_blad\_kmb = blad\_kmb;  }  dane\_blad\_tmax << log10( h ) << " " << log10( max\_blad\_lassonen ) << " " <<  log10( max\_blad\_kmb ) << endl;  //usuwanie tablic z pamieci  for ( int i = 0; i < t\_size\_kmb; i++ ){  delete[] Analityczne\_K[i];  delete[] KMB[i];  }  for ( int i = 0; i < t\_size\_laasonen; i++ ){  delete[] Analityczne\_L[i];  delete[] Laasonen[i];  }  delete[] Analityczne\_K;  delete[] Analityczne\_L;  delete[] Laasonen;  delete[] KMB;  }  }  int main()  {  rozwiazanie();  system( "Pause" );  return 0;  } |