Sprawozdanie

Laboratorium komputerowe z przedmiotu „Metody Obliczeniowe”

Prowadzący: dr hab. Inż. L. Bieniasz

Ćwiczenie 11-3

Filip Merta

Nr albumu: 121072

Gr. Lab. 02

Temat ćwiczenia

Zagadnienie z warunkiem początkowym i brzegowym obejmuje:

Równanie różniczkowe cząstkowe: $\frac{\partial U(x,t)}{\partial t} = D[\frac{\partial^2 U(x,t)}{\partial x^2} + \frac{2}{x}\frac{\partial U(x,t)}{\partial x}]$, określone dla współrzędnej przestrzennej x ∈ [r,+∞] oraz czasu t ∈ [0,t_{max}],

Warunek początkowy U(x,0) = 1, oraz

Warunki brzegowe U(r,t)=0, U(+∞,t) = 1.

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

Rozwiązanie analityczne tego zagadnienia ma postać: $U(x,t) = 1 - \frac{r}{x} erfc(\frac{x-r}{2\sqrt{Dt}})$, gdzie erfc(z) = 1 - erf(z), a erf(z) jest tzw. Funkcją błędu: $\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}}\int_0^z \exp(-w^2)\, dw$.

Do obliczeń numerycznych przedział nieskończony x należy zastąpić przedziałem skończonym [r,r+a], a drugi warunek brzegowy zastąpić warunkiem $U(r+a,t) = 1 - \frac{r}{r+a} erfc(\frac{a}{2\sqrt{Dt}})$. Do obliczeń funkcji erfc(z) z dokładnością bliską dokładności maszynowej dla zmiennych typu double należy zastosować pakiet CALERF udostępniony przez prowadzącego zajęcia.

Należy rozwiązać to zagadnienie stosując zaznaczoną niżej kombinację algorytmów numerycznych oraz podane wartości parametrów. Należy przyjąć ustaloną wartość $\lambda = D\frac{\partial t}{h^2}$, możliwie najbliższą $\lambda = 0.4$ dla metody bezpośredniej lub $\lambda = 1$ dla metod pośrednich.

Do zaliczenia projektu należy wykonać:

(1) Wykres zależności maksymalnej wartości bezwzględnej błędu obserwowanej dla t_{max} w funkcji kroku przestrzennego h(najlepiej 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.

Algorytmy:

Dyskretyzacja:

-Klasyczna metoda bezpośrednia

-Metoda pośrednia Lassonen

Rozwiązanie algebraicznych układów równań liniowych:

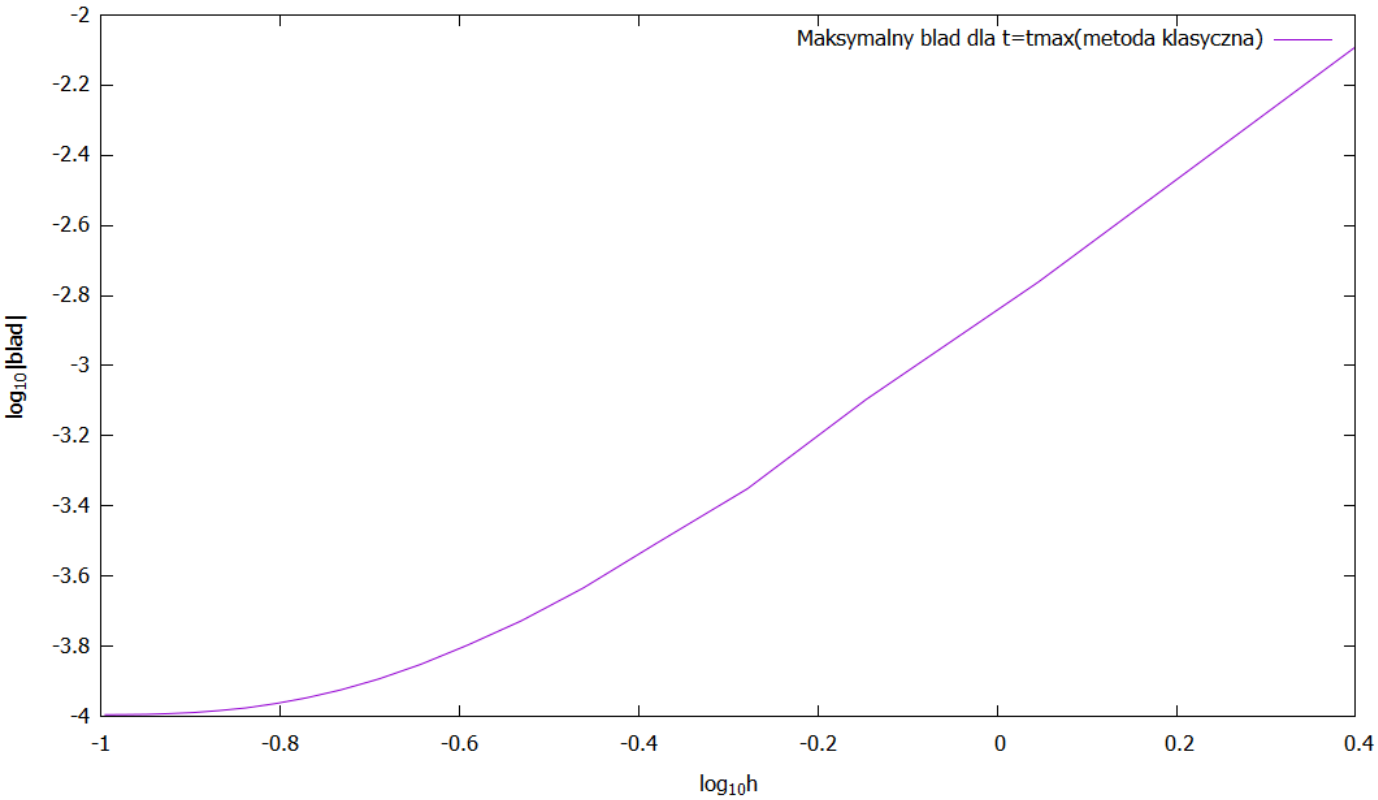
-Algorytm Thomasa

Parametry:

t_{max} = 2, r = 1, a = 10, D = 1.

Wykresy zależności maksymalnej wartości bezwzględnej błędu obserwowanej dla t_{max} w funkcji kroku przestrzennego h

Klasyczna metoda bezpośrednia



Rząd dokładności obliczony został na podstawie lokalnego błędu obcięcia danego wzorem:

$T = Ah^p$, gdzie p jest rzędem dokładności

Logarytmując obie strony oraz korzystając z własności logarytmów otrzymujemy równanie prostej:

$$log_{10}T = p * log_{10}h + log_{10}A$$

Rząd dokładności p jest w tym równaniu współczynnikiem kierunkowym który można obliczyć ze wzoru:

$p = \tan \propto$,gdzie $\propto$ jest kątem nachylenia prostej względem osi ox

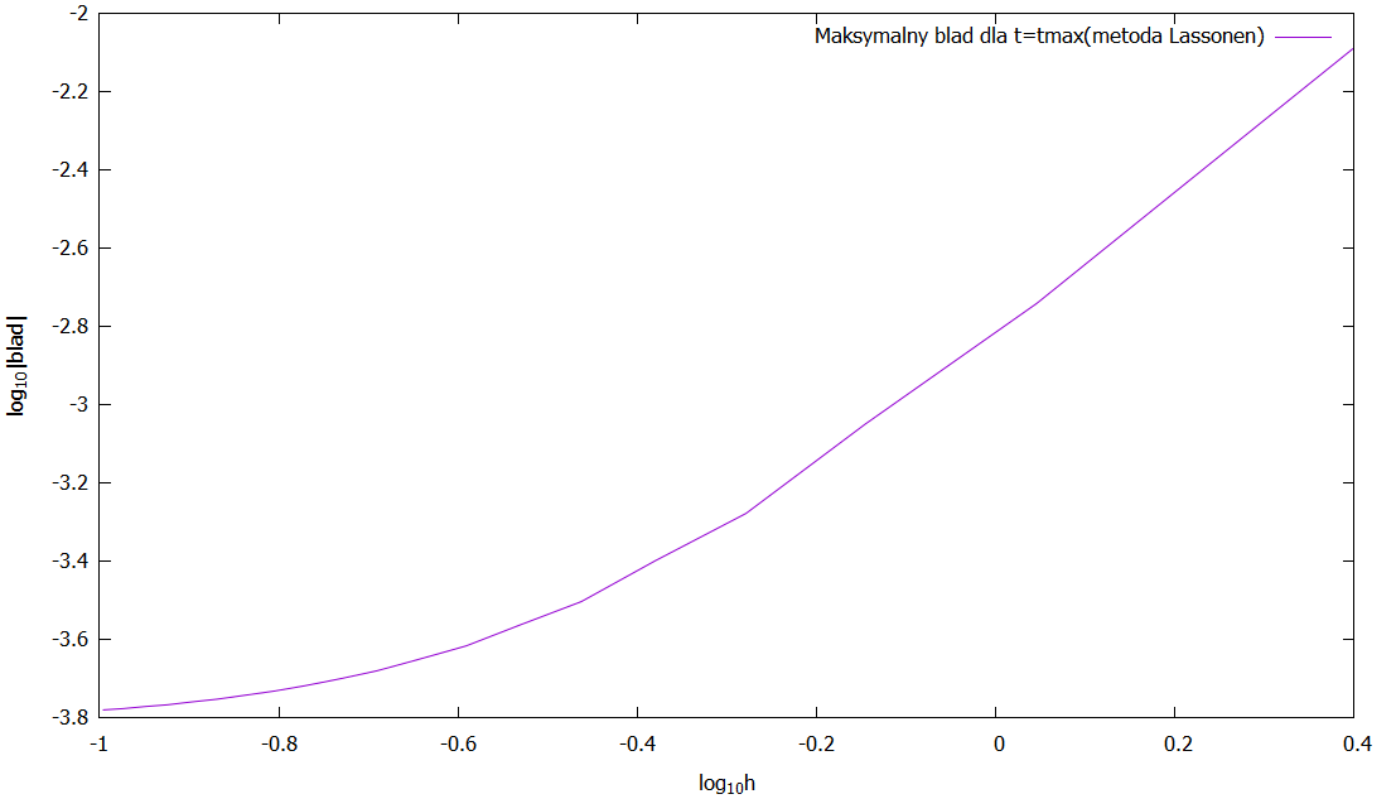
$$\tan \propto = \frac{różnica\ wartości\ funkcji\ dwóch\ kolejnych\ punktów}{różnica\ tych\ dwóch\ punktow}$$

Z powyższego wykresu program obliczył rząd dokładności:

$$p \approx 1.894 \approx 2$$

Co jest zgodne z teoretycznym rzędem dokładności

Metoda Lassonen



Rząd dokładności obliczony został na podstawie lokalnego błędu obcięcia danego wzorem:

$T = Ah^p$, gdzie p jest rzędem dokładności

Logarytmując obie strony oraz korzystając z własności logarytmów otrzymujemy równanie prostej:

$$log_{10}T = p * log_{10}h + log_{10}A$$

Rząd dokładności p jest w tym równaniu współczynnikiem kierunkowym który można obliczyć ze wzoru:

$p = \tan \propto$,gdzie $\propto$ jest kątem nachylenia prostej względem osi ox

$$\tan \propto = \frac{różnica\ wartości\ funkcji\ dwóch\ kolejnych\ punktów}{różnica\ tych\ dwóch\ punktow}$$

Z powyższego wykresu program obliczył rząd dokładności:

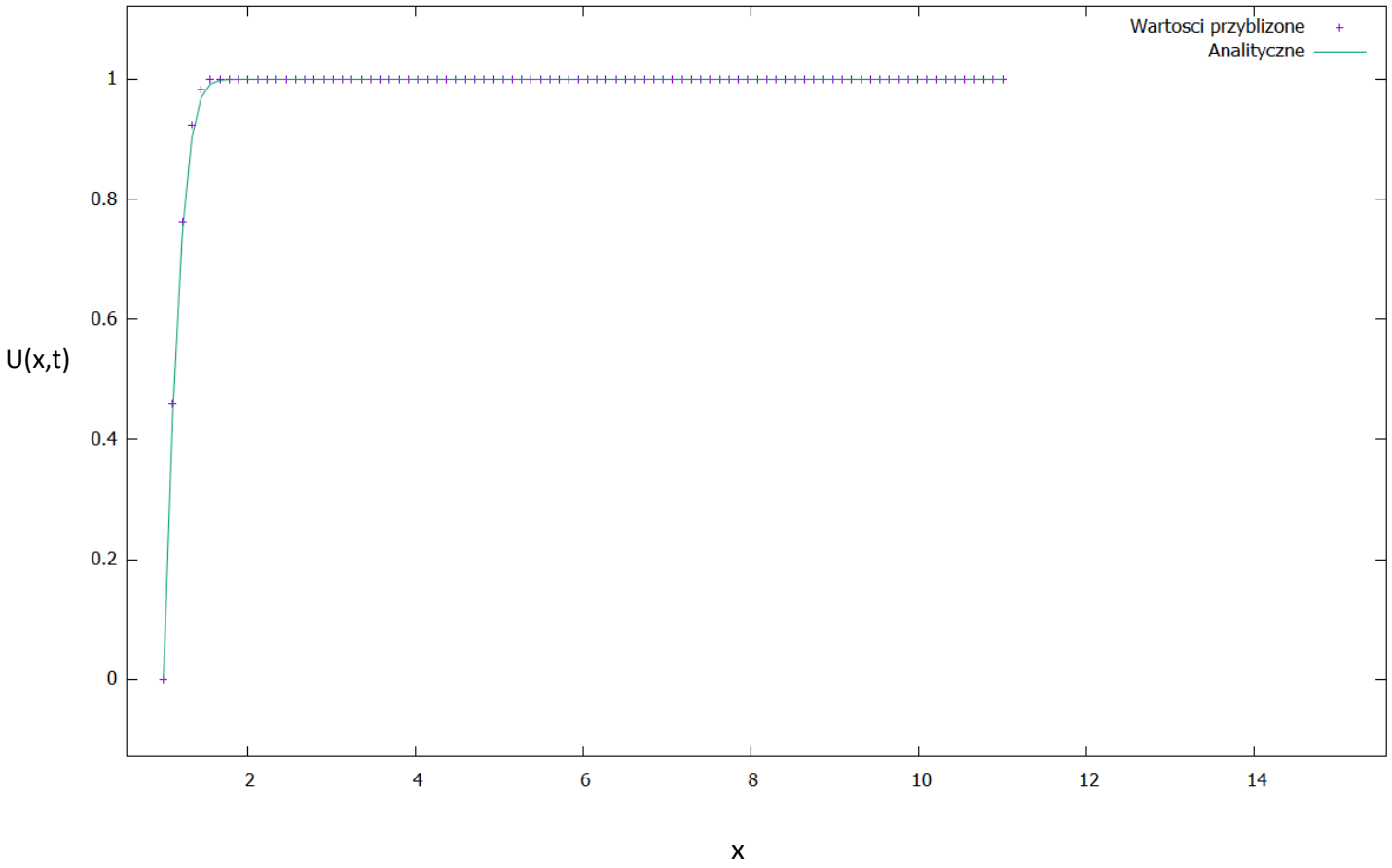
$p \approx 1.845 \approx 2$

Co jest zgodne z teoretycznym rzędem dokładności

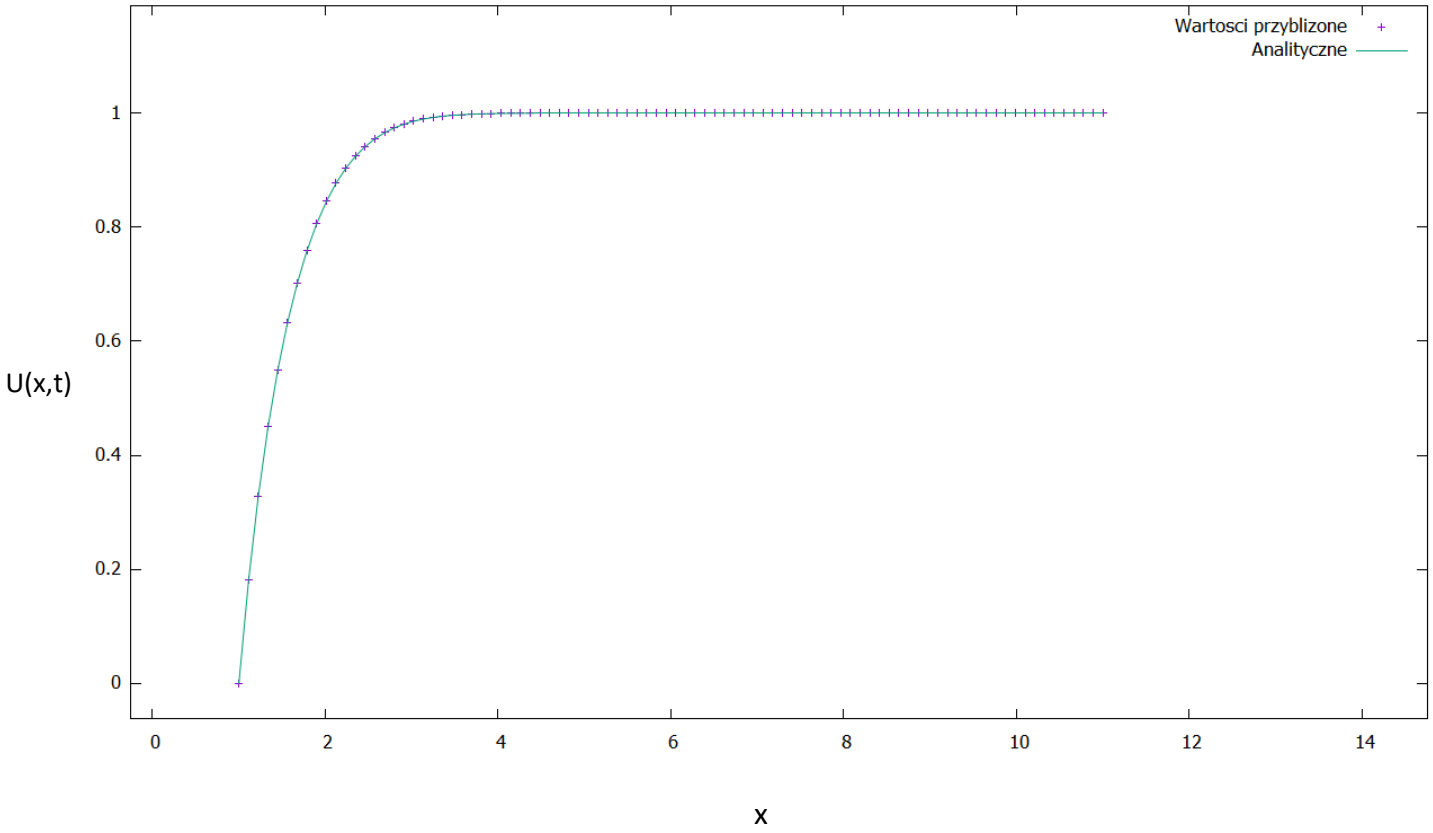
Wykresy rozwiązań numerycznych i analitycznych dla kilku wybranych wartości czasu t z całego przedziału t

Klasyczna metoda bezpośrednia($\lambda = 0.397043$)

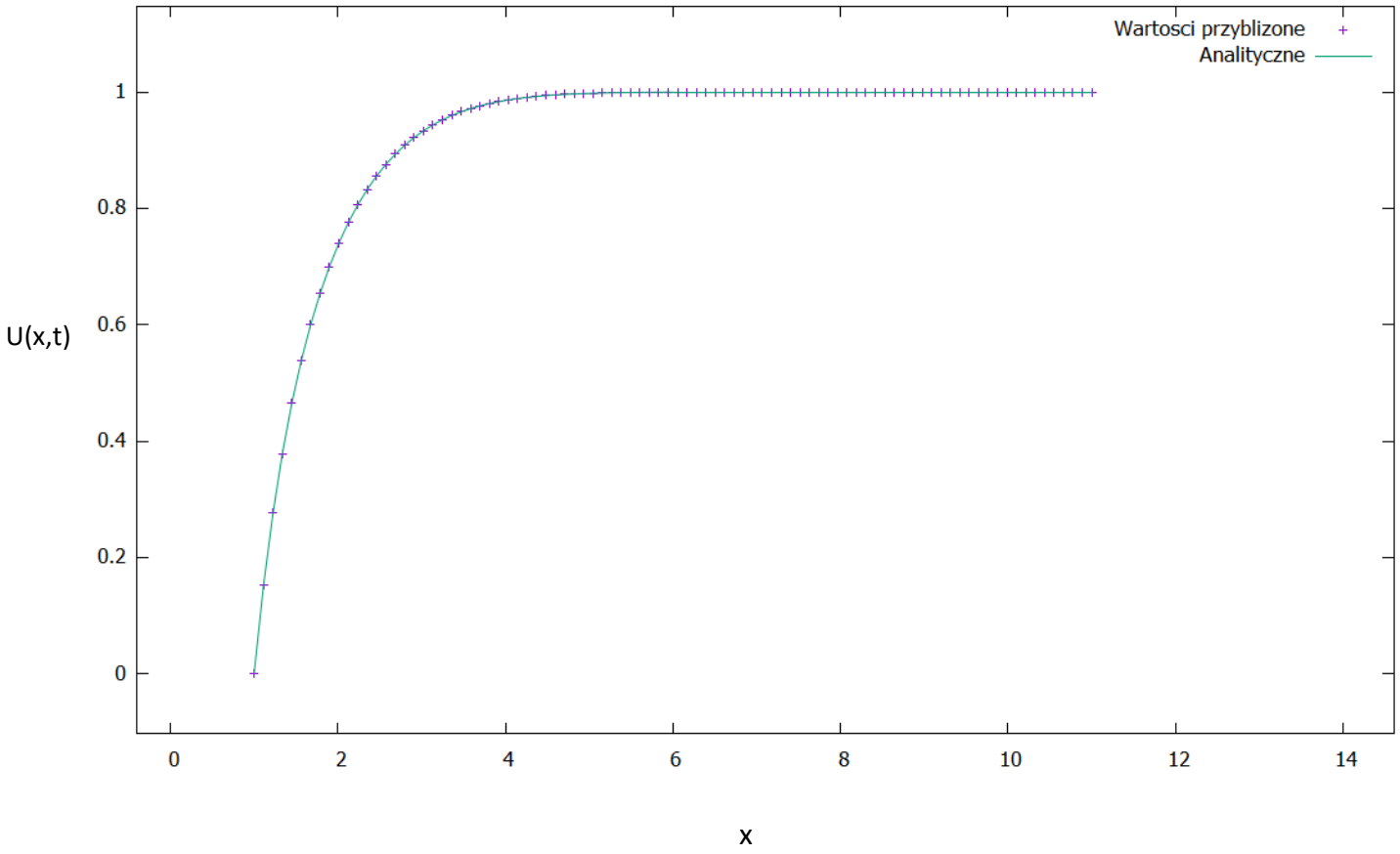
t = 0.025062656641604



t = 0.501253132832080

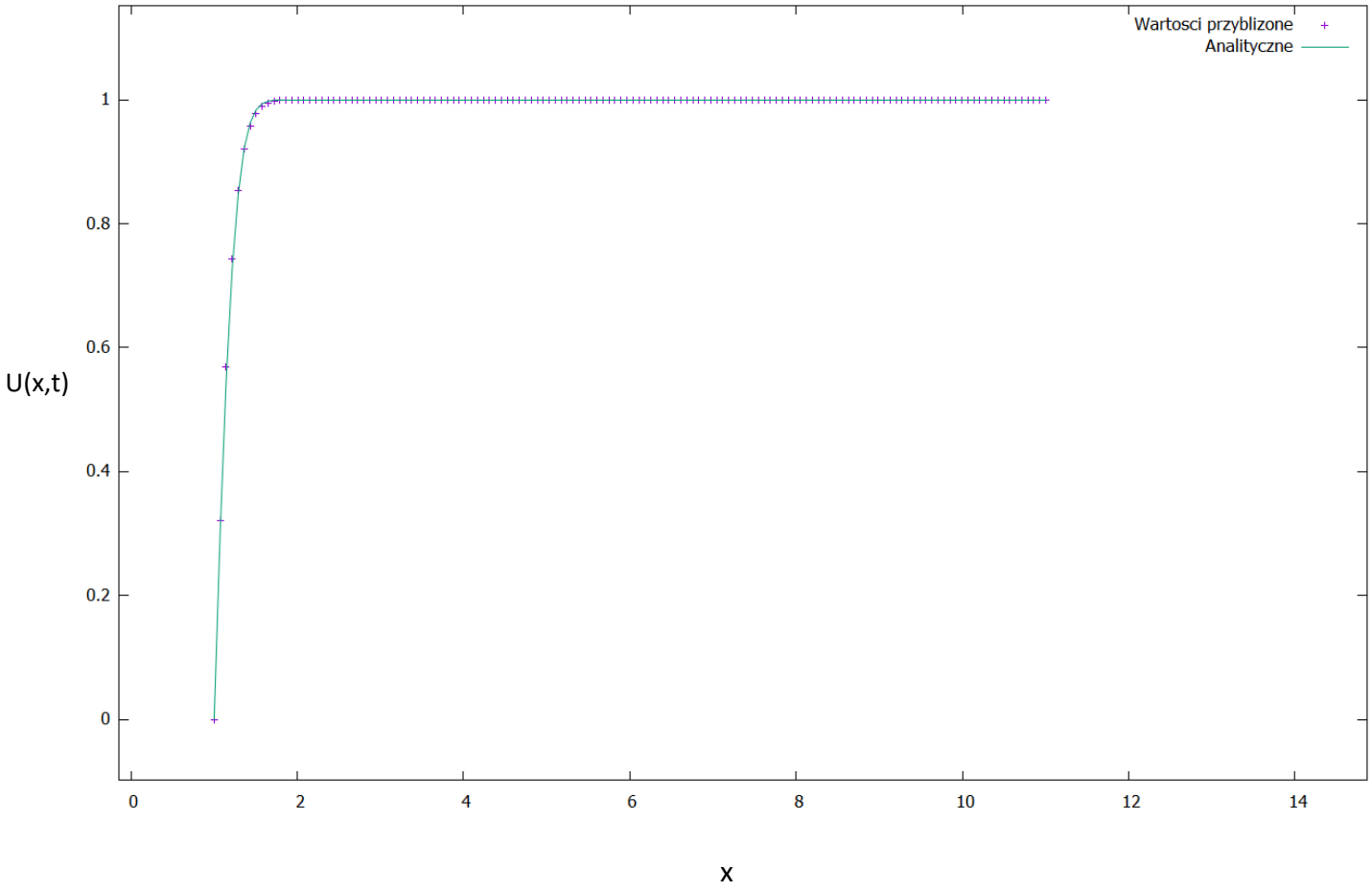


t = 1.253132832080200

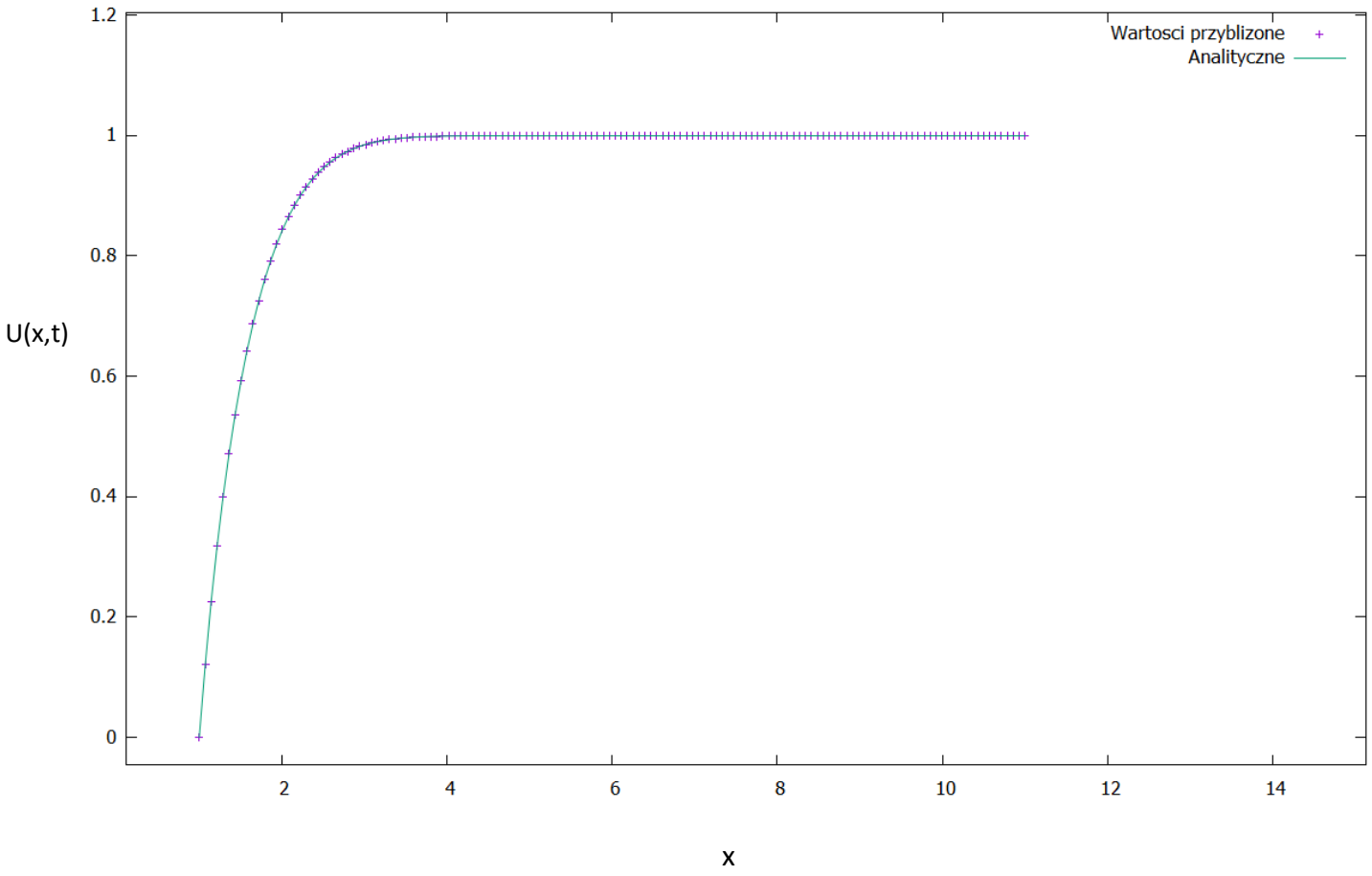


Metoda Lassonen($\lambda = 0.968471$)

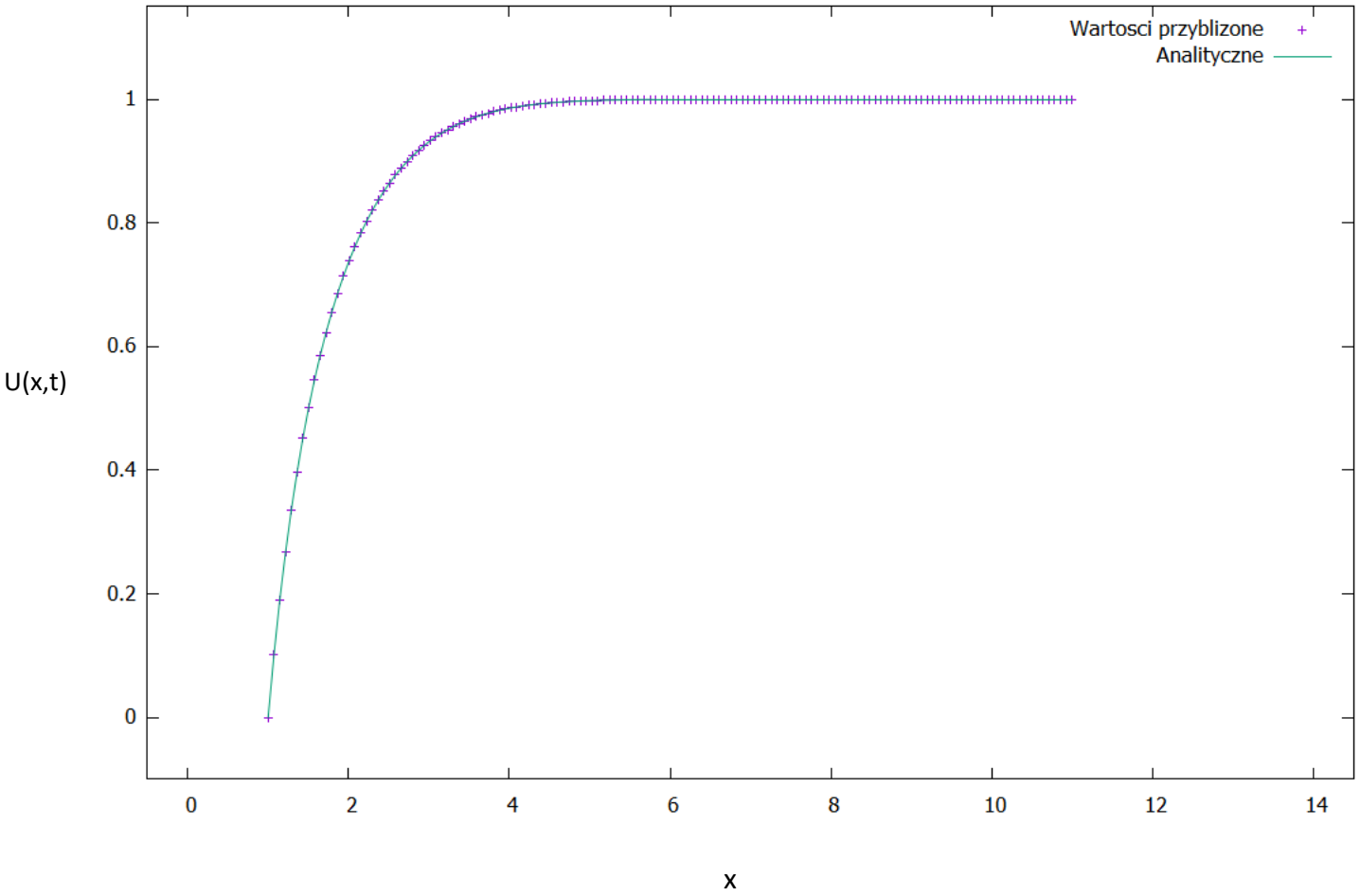
t = 0.025062656641604



t = 0.501253132832080



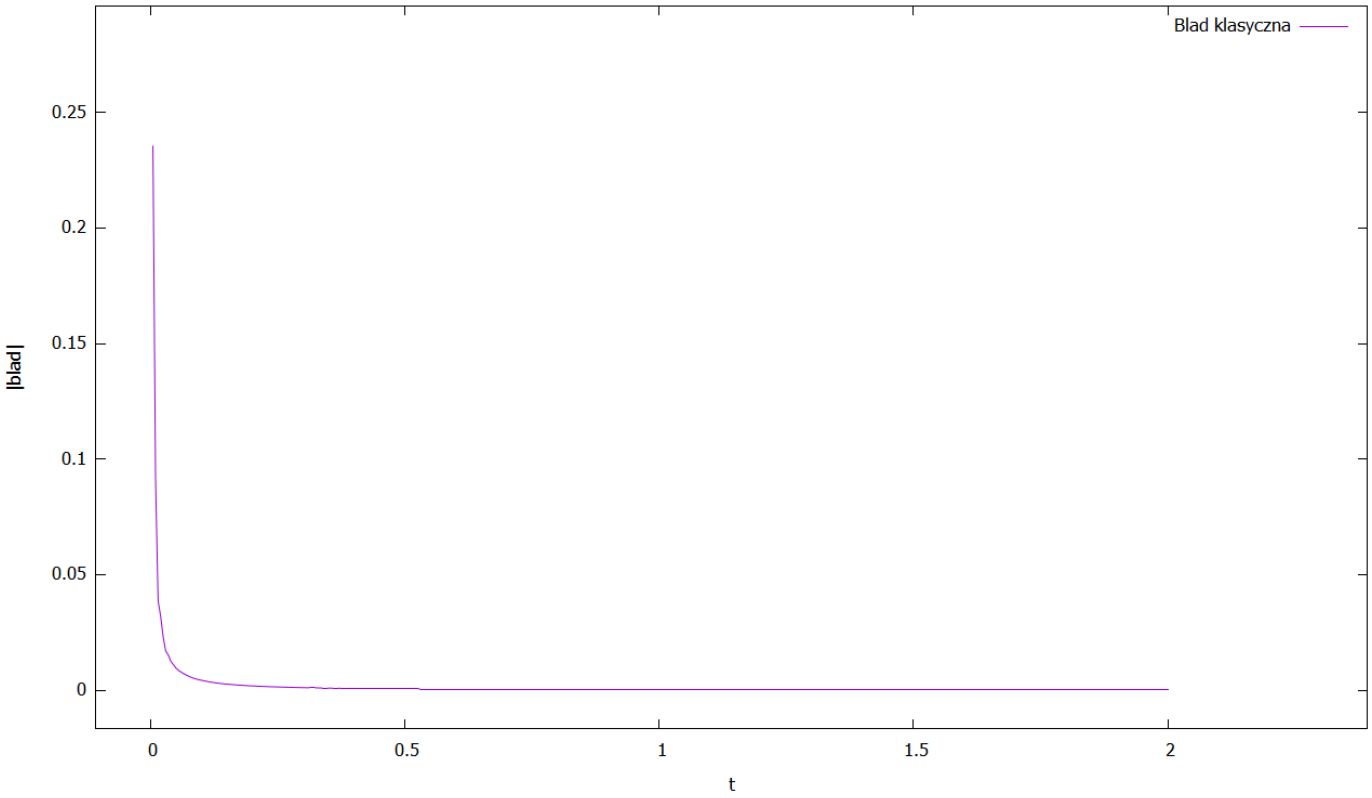
t = 1.253132832080200



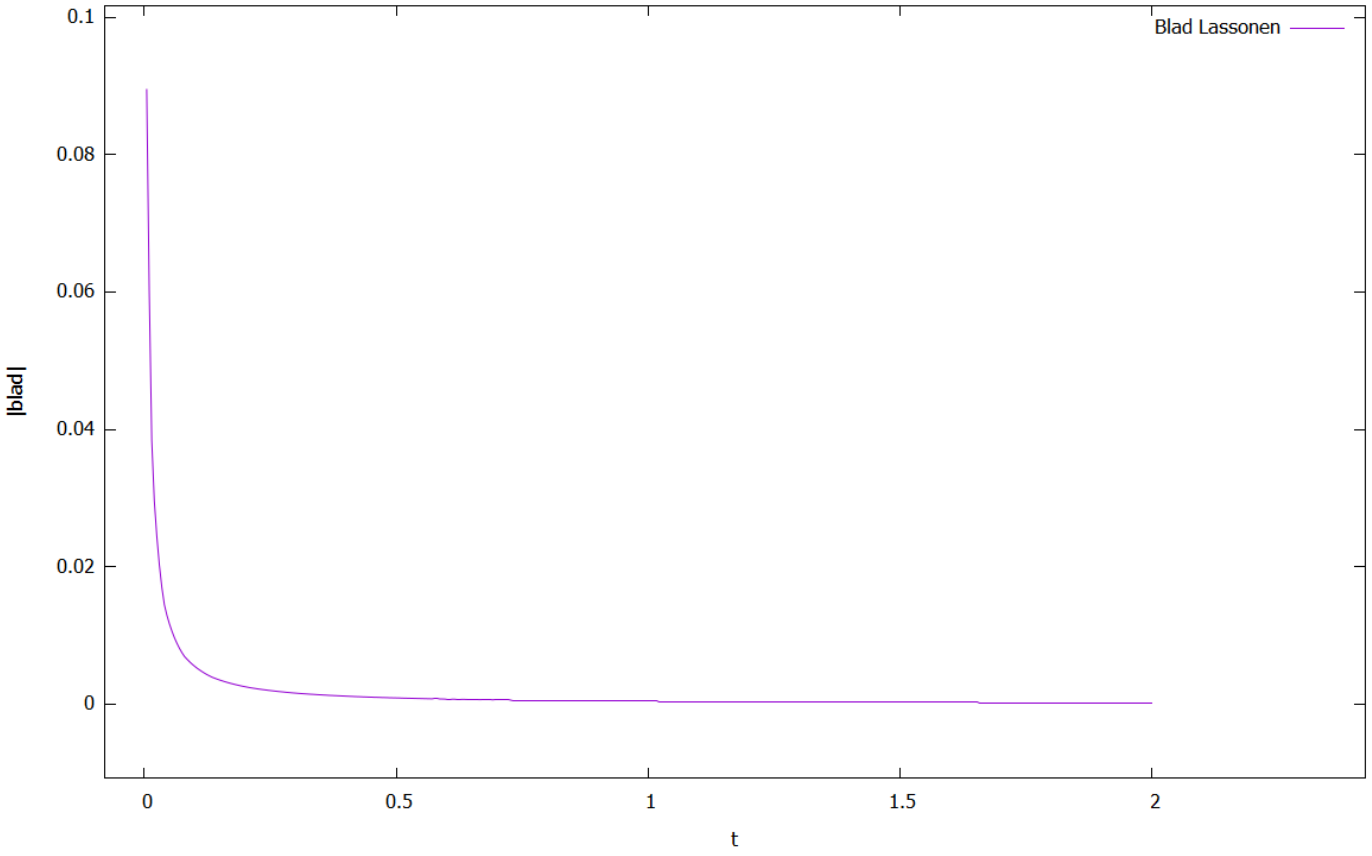
Widać że obie metody dobrze przybliżają wartości analityczne funkcji zwłaszcza dla większych kroków czasowych. Różnice pomiędzy analitycznym rozwiązaniem, a numerycznym widać przy kroku czasowym t = 0.025062656641604 w miejscu gdzie funkcja bardzo szybko rośnie. Widać, że metoda Lassonen lepiej radzi sobie z tym przypadkiem.

Wykresy zależności maksymalnej wartości bezwzględnej błędu w funkcji czasu t

Klasyczna metoda bezpośrednia($\lambda = 0.397043$)



Metoda Lassonen($\lambda = 0.968471$)



Na wykresach widać że obie metody na początku mają dość duży błąd lecz z każdym kolejnym krokiem błąd maleje. Co więcej metoda Lassonen ma na początku mniejszy błąd. Zaczyna z błędem w okolicach 0.09, gdzie klasyczna metoda bezpośrednia zaczyna z błędem bliskim 0.25. Obie metody są stabilne ponieważ błąd nie rośnie więc błąd każdego kroku musi być w następnym kroku tłumiony. Początkowa duża wartość błędu może być spowodowana wartościami kolejnych pochodnych które wpływają na lokalny błąd obcięcia.

Kod źródłowy:

```
#include "stdafx.h"
#define _USE_MATH_DEFINES
#include <iostream>
#include <iomanip>
#include <cmath>
#include <fstream>
#include "calerf.h"


using namespace std;

//Zmienne globalne
const long double tmax = 2.0L;
const long double r = 1.0L;
const long double a = 10.0L;
const long double D = 1.0L;

//Funkcja odpowiedzalna na ustawienie współczynników eta na przekątnej D w algorytmie thomasa
void eta(long double *D, long double *L, long double *U, int N) {
	for (int i = 1; i < N; i++) D[i] = D[i] - (L[i - 1] / D[i - 1]) * U[i - 1];
}

//Funkcja odpowiedzialna za ustawienie wektora b oraz obliczenie rozwiązania x w algorytmie thomasa
void rr(long double *b, long double *L, long double *D, long double *U, long double *x, int N) {
	int i;
	for (i = 1; i <= N; i++) b[i] = b[i] - (L[i - 1] / D[i - 1])*b[i - 1];
	x[N-1] = b[N-1] / D[N-1];
	for (i = N - 2; i >= 0; i--) x[i] = ((b[i] - (x[i + 1] * U[i])) / D[i]);
}

//Algorytm thomasa
void thomas(long double *L, long double *D, long double *U, long double *x, long double *b, int N) {
	eta(D, L, U, N);
	rr(b, L, D, U, x, N);
}

//Funkcja obliczająca analityczne rozwiązanie szukanej funkcji w punkcie x,t
long double analityczne(const long double x, const long double t) {
	return 1.0 - (r / x)*ERFCL((x - r) / (2.0*sqrt(D*t)));
}

//Funkcje obliczające wartość współczynników przy pochodnych cząstkowych w punkcie x
long double ax(const long double x) { return D; }
long double dx(const long double x) { return D*(2.0L / x); }
long double ex(const long double x) { return 1.0L; }

//Funkcja odpowiedzialna za alokacje tablicy w której zostaną przechowane obliczone wartości szukanej funkcji w odpowiednich punktach
void alokuj(long double ***A,const int n, const int m) {
	*A = new long double*[n];
	for (int i = 0; i < n; i++) (*A)[i] = new long double[m];
}

//Funkcja odpowiedzialna za zwolnienie pamięci po wykonanych obliczeniach
void zwolnij_pamiec(long double **A,const int n) {
```

```
	for (int i = 0; i < n; i++) delete[] A[i];
	delete[] A;
}

//Funkcja odpowiedzialna za alokacje wektorów potrzebnych do algorytmu tomasa
void alokujLDU(long double **a,long double **b,long double **c, long double **d,const int m) {
	*a = new long double[m-1];
	*b = new long double[m];
	*c = new long double[m - 1];
	*d = new long double[m];
}

//Funkcja odpowiedzialna za zwolnienie pamięci po wektorach używanych do algorytmu thomasa
void zwolnij_pamiecLDU(long double **a, long double **b, long double **c, long double **d) {
	delete[] a;
	delete[] b;
	delete[] c;
	delete[] d;
}


//Funkcja odpowiedzialna za obliczanie kolejnych wartości szukanej funkcji klasyczną metodą bezpośrednią
//w kolejnych krokach czasowych i zapisująca wyniki
//w tablicy dwuwymiarowej. Dla każdego kroku czasowego i obliczam wartości funkcji w punktach xj korzystając z wartości
//funkcji w kroku czasowym i-1 poprzez trzypunktowe przybliżenie centralne na 2 pochodną oraz metode bezpośrednią Eulera.
void metoda_klasyczna(long double **A, const int n, const int m,const long double h, const long double dt ) {
	long double lambda = D * (dt / (h*h));
	cout << lambda << " Klasyczna" << endl;
	for (int i = 1; i < n; i++)
		for (int j = 1; j < (m - 1); j++) A[i][j] = lambda * A[i - 1][j - 1] + (1.0L - 2.0L*lambda)*A[i - 1][j] + lambda * A[i - 1][j
+ 1] + (D*(dt/(2.0L*h)))* A[i - 1][j + 1] * (2.0L / (r + (long double)(j) * h)) - (D*(dt / (2.0L*h))) * A[i - 1][j - 1] * (2.0L / (r + (long
double)(j) * h));
}


//Funkcja odpowiedzialna za obliczanie kolejnych wartości szukanej funkcji metodą Lassonen. Odbywa się to poprzez
//stworzenie układu równań różnicowych w których korzystam z trzypunktowego przybliżenia centralnego na 2 pochodną
//i pośredniej metody Eulera. W układzie równań Ax=b, A jest macierza trójdiagonalną więc korzystam z algorytmu thomasa
//do obliczenia wartości funkcji.
void metoda_lassonen(long double **U, long double *LL, long double *DD, long double *UU, long double *b, const int n, const
int m, const long double h, const long double dt) {
	long double lambda = D * (dt / (h*h));
	cout << lambda << endl;
	for (int i = 1; i < n; i++) {
		b[0] = U[i][0];
		b[m - 1] = U[i][m - 1];
		for (int j = 1; j < (m - 1); j++) b[j] = -U[i - 1][j];
		DD[0] = 1.0L;
		DD[m - 1] = 1.0L;
		for (int j = 1; j < m - 1; j++) DD[j] = -1.0L*(1.0L + 2.0L*lambda);
		LL[m - 2] = 0.0L;
		for (int j = 0; j < m - 2; j++) LL[j] = lambda - D * dt/((r + (long double)(j+1)* h)*h);
		//for (int j = 0; j < m - 2; j++) LL[j] = lambda;
		UU[0] = 0.0L;
		for (int j = 1; j < m - 1; j++) UU[j] = lambda + D * dt/((r + (long double)(j)* h)*h);
		//for (int j = 1; j < m - 1; j++) UU[j] = lambda;
		thomas(LL, DD, UU, U[i], b, m);
	}
}

//Funkcja obliczająca wartość bezwzględną z największego błędu przybliżonego rozwiązania z rozwiązaniem analitycznym przy
//kroku czasowym t
//Norma maksimum wektora
```

```
long double maxblad(long double *A, int n, int m,long double h,long double t) {
	long double maxblad = 0.0L;
	long double blad = 0.0L;
	for (int j = 0; j < m; j++) {
		blad = fabsl(analityczne(r+j*h,t) - A[j]);
		if (blad > maxblad) maxblad = blad;
	}
	return maxblad;
}

//Funkcja obliczająca wartość bezwzględną z największego błędu z całej tablicy przybliżonych rozwiązań
long double maxblad(long double **A, int n, int m, long double h, long double t) {
	long double maxblad = 0.0L;
	long double blad = 0.0L;
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < m; j++) {
			blad = fabsl(analityczne(r + (long double)j * h, 0.0L+(long double)i*t) - A[i][j]);
			if (blad > maxblad) maxblad = blad;
		}
	}
	return maxblad;
}

int main()
{
	//Inicjalizacja zmiennych
	int t;
	int x;
	long double blad=0.0L;
	long double last_blad;
	long double last_h;
	long double analit;
	long double **U;
	long double *LL;
	long double *DD;
	long double *UU;
	long double *b;
	long double h;
	long double dt;
	ofstream fs2;

	// Metoda klasyczna
	// Ilości wezłów dla czasu t i punktów x
	t = 400;
	x = 90;
	// Obliczanie kroku h oraz korku dt na podstawie ilości wezłów
	h = (r + a - r) / ((long double)x - 1.0L);
	dt = tmax / ((long double)t - 1.0L);
	//Alokacja odpowiedniej tabliy dla wybranych ilości wezłów
	alokuj(&U, t, x);
	//Ustawienie warunku początkowego i warunków brzegowych
	for (int i = 0; i < x; i++) { U[0][i] = 1.0L; }
	for (int i = 1; i < t; i++) {
		U[i][0] = 0.0L;
		U[i][x-1] = 1.0L -(r/(r+a))*ERFCL(a/(2.0L*(long double)sqrt(D*(double)i*dt)));
	}
	//Uruchomienie metody odpowiedzialnej za obliczenie przybliżonych wartości szukanej funkcji
	metoda_klasyczna(U, t, x,h,dt);
	//Zapis wyników do pliku(wartosci x, wartosci t, rozwiazanie przyblizone, rozwiazanie analityczne i błąd)
	fs2 << setprecision(15);
	fs2.open("met_klasyczna.dat", fstream::out);
	fs2 << setw(25) << left << "x"<<'\t';
	fs2 << setw(25) << left << "t" << '\t';
```

```
	fs2 << setw(25) << left << "Ux,t" << '\t';
	fs2 << setw(25) << left << "Analityczne" << '\t';
	fs2 << setw(25) << left << "Blad" << endl;
	for (int i = 0; i < t; i++) {
		for (int j = 0; j < x; j++) {
			analit = analityczne(r + (long double)j*h, 0.0L + (long double)i*dt);
			fs2 << setw(25) << left << r + (long double)j*h << '\t' << setw(25) << left << 0.0L + (long double)i*dt;
			fs2 << '\t' << setw(25) << left << U[i][j] << '\t' << setw(25) << left << analit;
			fs2 << scientific;
			fs2 << '\t' << setw(25) << left << abs(U[i][j]-analit)<<endl;
			fs2 << fixed;
		}
	}
	fs2 << scientific;
	fs2 << setw(25) << left << maxblad(U,t,x,h,dt);
	fs2 << fixed;
	fs2.close();
	//Obliczenie maksymalnego bezwzględnego błędu dla każdego kroku czasowego t z zapisem do pliku
	fs2.open("bladt_klasyczna.dat", fstream::out);
	for (int i = 1; i < t; i++) fs2 << setw(25) << left << 0.0L + i * dt << '\t' << setw(25) << left << maxblad(U[i], t, x, h, 0.0L+i*dt)
<< endl;
	fs2.close();
	//Zapis do pliku wartości funkcji uzyskanej i wartości analityczne dla 3 różnych chwili czasu t
	fs2.open("met_klasycznawyn.dat", fstream::out);
	fs2 << "#t1 = " << 0.0L + 5 * dt << endl;
	fs2 << "#t2 = " << 0.0L + 100 * dt << endl;
	fs2 << "#t3 = " << 0.0L + 250 * dt << endl;
	for (int i = 0; i < x; i++) {
		fs2 << setw(25) << left << r + i * h << '\t' << setw(25) << left << U[5][i] << '\t' << setw(25) << left << analityczne(r
+ i * h,0.0L+5*dt) << '\t';
		fs2 << '\t' << setw(25) << left << U[100][i] << '\t' << setw(25) << left << analityczne(r + i * h, 0.0L + 100 * dt) <<
'\t';
		fs2 <<  '\t' << setw(25) << left << U[250][i] << setw(25) << left << analityczne(r + i * h, 0.0L + 250 * dt) <<endl;
	}
	fs2.close();
	//zwolnienie pamięci
	zwolnij_pamiec(U, t);

	// Metoda Lassonen
	//Ustawienie ilości węzłów
	t = 400;
	x = 140;
	//Obliczenie h oraz dt na podstawie ilości węzłów
	h = (r + a - r) / ((long double)x - 1.0L);
	dt = tmax / ((long double)t - 1.0L);
	//Alokacja pamięci dla przybliżonych rozwiązań
	alokuj(&U, t, x);
	//Alokacja pamięci dla algorytmu thomasa
	alokujLDU(&LL, &DD, &UU, &b, x);
	//Ustawienie warunku początkowego i warunków brzegowych
	for (int i = 0; i < x; i++) { U[0][i] = 1.0L; }
	for (int i = 1; i < t; i++) {
		U[i][0] = 0.0L;
		U[i][x - 1] = 1.0L - (r / (r + a))*ERFCL(a / (2.0L*(long double)sqrt(D*(double)i*dt)));
	}
	//Uruchomienie algorytmu obliczającego metodą Lassonen
	metoda_lassonen(U,LL,DD,UU,b ,t, x, h, dt);
	//Zapis wyników do pliku
	fs2 << setprecision(15);
	fs2.open("met_lassonen.dat", fstream::out);
	fs2 << setw(25) << left << "x" << '\t';
	fs2 << setw(25) << left << "t" << '\t';
	fs2 << setw(25) << left << "Ux,t" << '\t';
```

```
	fs2 << setw(25) << left << "Analityczne" << '\t';
	fs2 << setw(25) << left << "Blad" << endl;
	for (int i = 0; i < t; i++) {
		for (int j = 0; j < x; j++) {
			analit = analityczne(r + (long double)j*h, 0.0L + (long double)i*dt);
			fs2 << setw(25) << left << r + (long double)j*h << '\t' << setw(25) << left << 0.0L + (long double)i*dt;
			fs2 << '\t' << setw(25) << left << U[i][j] << '\t' << setw(25) << left << analit;
			fs2 << scientific;
			fs2 << '\t' << setw(25) << left << abs(U[i][j] - analit) << endl;
			fs2 << fixed;
		}
	}
	fs2 << scientific;
	fs2 << setw(25) << left << maxblad(U, t, x, h, dt);
	fs2 << fixed;
	fs2.close();
	//Obliczenie maksymalnego bezwzględnego błędu dla każdego kroku czasowego t z zapisem do pliku
	fs2.open("bladt_lassonen.dat", fstream::out);
	for (int i = 1; i < t; i++) fs2 << setw(25) << left << 0.0L + i * dt << '\t' << setw(25) << left << maxblad(U[i], t, x, h, 0.0L + i *
dt) << endl;
	fs2.close();
	//Zapis do pliku wartości funkcji uzyskanej i wartości analityczne dla 3 różnych chwili czasu t
	fs2.open("met_lassonenwyn.dat", fstream::out);
	fs2 << "#t1 = " << 0.0L + 5 * dt << endl;
	fs2 << "#t2 = " << 0.0L + 100 * dt << endl;
	fs2 << "#t3 = " << 0.0L + 250 * dt << endl;
	for (int i = 0; i < x; i++) {
		fs2 << setw(25) << left << r + i * h << '\t' << setw(25) << left << U[5][i] << '\t' << setw(25) << left << analityczne(r
+ i * h, 0.0L + 5 * dt) << '\t';
		fs2 <<  '\t' << setw(25) << left << U[100][i] << '\t' << setw(25) << left << analityczne(r + i * h, 0.0L + 100 * dt) <<
'\t';
		fs2 <<  '\t' << setw(25) << left << U[250][i] << setw(25) << left << analityczne(r + i * h, 0.0L + 250 * dt) << endl;
	}
	fs2.close();
	//Zwolnienie pamieci
	zwolnij_pamiec(U, t);


	//Obliczanie maksymalnego bezwzględnego błędu dla czasu tmax w zależności od kroku h dla metody Lassonen
	//zmienając ilość wezłów na osi x
	fs2 << setprecision(15);
	fs2.open("blad_lassonen.dat", fstream::out);
	for (int i = 100; i > 0; i = i - 5) {
		t = 400;
		x = i;
		last_blad = blad;
		last_h = h;
		h = (r + a - r) / ((long double)x - 1.0L);
		dt = tmax / ((long double)t - 1.0L);
		alokuj(&U, t, x);
		alokujLDU(&LL, &DD, &UU, &b, x);
		for (int i = 0; i < x; i++) { U[0][i] = 1.0L; }
		for (int i = 1; i < t; i++) {
			U[i][0] = 0.0L;
			U[i][x - 1] = 1.0L - (r / (r + a))*ERFCL(a / (2.0L*(long double)sqrt(D*(double)i*dt)));
		}
		metoda_lassonen(U, LL, DD, UU, b, t, x, h, dt);

		blad = maxblad(U[t-1], t, x, h, 0.0L+(t-1)*dt);
		fs2 << setw(15) << left << log10(h) <<'\t'<< setw(15) << left << log10(blad) << endl;

		zwolnij_pamiec(U, t);
```

```
	}
	//Obliczanie rzędu dokładności metody Lassonen
	cout <<"Rzad dokladnosci h metoda Lassonen: "<< ((log10(blad) - log10(last_blad))/(log10(h)-log10(last_h)))<<endl;
	fs2.close();

	//Obliczanie maksymalnego bezwzględnego błędu dla czasu tmax w zależności od kroku h dla bezpośredniej metody klasycznej.
	//Uruchamiam algorytm zmieniając ilość węzłów na osi x

	fs2.open("blad_klasyczna.dat", fstream::out);
	for (int i = 100; i > 0; i = i - 5) {
		t = 400;
		x = i;
		last_blad = blad;
		last_h = h;
		h = (r + a - r) / ((long double)x - 1.0L);
		dt = tmax / ((long double)t - 1.0L);
		alokuj(&U, t, x);
		alokujLDU(&LL, &DD, &UU, &b, x);
		for (int i = 0; i < x; i++) { U[0][i] = 1.0L; }
		for (int i = 1; i < t; i++) {
			U[i][0] = 0.0L;
			U[i][x - 1] = 1.0L - (r / (r + a))*ERFCL(a / (2.0L*(long double)sqrt(D*(double)i*dt)));
		}
		metoda_klasyczna(U, t, x, h, dt);

		blad = maxblad(U[t - 1], t, x, h, 0.0L + (t - 1)*dt);
		fs2 << setw(15) << left << log10(h) << '\t' << setw(15) << left << log10(blad) << endl;

		zwolnij_pamiec(U, t);
	}
	//Obliczanie rzędu dokładności klasycznej metody bezpośredniej
	cout << "Rzad dokladnosci h metoda klasyczna: " << ((log10(blad) - log10(last_blad)) / (log10(h) - log10(last_h))) << endl;
	fs2.close();


   return 0;
}
```