```
#include <iostream>
#include <fstream>
#include <cmath>
#include <cstdlib>
#include <complex>

using namespace std;

int main()
{
    ofstream fout("difrakcia.dat");
    if(!fout)
    {
        cout << "Neda sa zapisovat do suboru difrakcia.dat" <<
             endl;
        return 1;
    }
    double L=1;
    double d=60e-6;
    double lambda=600e-9;
    double k=2*M_PI/lambda;
    double ymin=-0.05;
    double ymax=0.05;
    double step=(ymax-ymin)/500;
    for(double y=ymin;y<=ymax;y+=step)
    {
        complex <double> amplituda(0,0);
        for(double yy=-d/2;yy<=d/2;yy+=10*lambda)
        {
            double r=sqrt(pow(L,2)+pow(y-yy,2));
            complex <double> faza(0,-k*r);
            amplituda+=exp(faza);
        }

        double intenzita = norm(amplituda);
        fout << y << "\t" << intenzita << endl;
        cout << y << "\t" << intenzita << endl;
    }
    fout.close();
    cout << "Udaje su ulozene v subore difrakcia.dat" << endl;
    system("wgnuplot -persist -e \"set style data lines; plot
          'difrakcia.dat'\"");
    return 0;
}
```

FRANTIŠEK KUNDRACIK

ZÁKLADY PROGRAMOVANIA PRAKTICKY

UNIVERZITA KOMENSKÉHO BRATISLAVA 2013

Autor: FRANTIŠEK KUNDRACIK

Názov: ZÁKLADY PROGRAMOVANIA PRAKTICKY

Recenzenti: doc. RNDr. Vladimír Černý, CSc.
doc. RNDr. Ľudovít Fischer, CSc.

Vydavateľ: Vydavateľstvo UK, Šafárikovo námestie 6, 818 06 Bratislava 16
Grafická úprava: František Kundracik

Rok vydania: 2013
Miesto vydania: Bratislava
Vydanie prvé
Počet strán: 115

ISBN 978-80-8147-011-0

9 788081 470110 >

FRANTIŠEK KUNDRACIK

ZÁKLADY PROGRAMOVANIA PRAKTICKY

UNIVERZITA KOMENSKÉHO BRATISLAVA 2013

Obsah

1. Úvod

Prácu fyzika si dnes nemožno predstaviť bez aspoň občasných numerických výpočtov na počítači. Na zvládnutie tohto cieľa nemusí byť fyzik špičkovým programátorom, musí byť však schopný samostatne zvládnuť aspoň základné potrebné úkony, ako nainštalovanie vhodného kompilátora a naprogramovanie niektorých numerických metód. K tomu má pomôcť učebnica "Základy programovania prakticky". Je určená najmä pre študentov 1. ročníka fyziky na FMFI UK ako materiál k cvičeniam k predmetu "Základy programovania". Jej cieľom je precvičiť si základné princípy programovania fyzikálnych úloh v jazyku C++. Keďže ide o úvod do programovacích metód, ťažisko sa kladie najmä na zvládnutie základnej syntaxe jazyka tak, aby čitateľ zvládol samostatne naprogramovať jednoduchšie úlohy. C++ síce umožňuje objektovo orientovaný prístup, úlohy v tejto učebnici sa však riešia zásadne iba procedurálnymi programovacími metódami. Objektovo orientovaný prístup nechávame na neskoršie pokročilejšie štúdium.

Učebnica je koncipovaná tak, že okrem získavania praxe v programovaní je jej ďalším cieľom oboznámiť študentov s niektorými jednoduchými numerickými metódami používanými vo fyzike, ako je napríklad numerické integrovanie, riešenie diferenciálnych rovníc Eulerovou metódou, vykreslenie jednoduchého grafu a podobne. Od tohto prístupu si sľubujeme najmä prepojenie výučby programovania s fyzikou, čo čitateľovi na jednej strane pomôže prehĺbiť si vedomosti z fyziky a na strane druhej veríme, že sa tým učebnica "Základy programovania prakticky" stane atraktívnejšou.

Obťažnosť úloh sa zvyšuje iba postupne. Na začiatku študenti pracujú s pripravenými programami, kde cieľom je naučiť sa čítať a modifikovať jednoduché cudzie programy. Táto zručnosť je často podceňovaná, ale znovupoužitie už hotových programov alebo ich častí je typickou črtou práce dnešných programátorov. Neskôr majú študenti k dispozícii kostry programov, do ktorých majú doprogramovať niektoré algoritmy. Mnohé sú k dispozícii aj vo forme vývojových diagramov. Cieľom je naučiť študentov myslieť algoritmicky. Hoci väčšina študentov absolvovala informatiku už na strednej škole, ukazuje sa, že značná časť študentov má s tým veľké problémy. Ku koncu už majú študenti za úlohu samostatne tvoriť jednoduché programy, pričom úlohy sú volené tak, že prirodzene nútia študentov používať vedomosti získané pri riešení predchádzajúcich úloh. Veríme, že takýto prístup uľahčí začiatky študentom, ktorí sa s reálnym programovaním doteraz nikdy nestretli.

Veľký dôraz sa kladie aj na samostatnú domácu prácu čitateľa, preto bez nainštalovania si vhodného kompilátora na domácom počítači alebo tablete nebude možné štúdium úspešne absolvovať. Cvičenia na FMFI UK sú vedené v integrovanom vývojovom prostredí CodeBlocks nainštalovanom v operačnom systéme Linux s rozhraním KDE. V domácom prostredí je najrozšírenejším operačným systémom MS Windows, prípadne počas cestovania OS Android na prenosných zariadeniach. Preto nasledujúca časť "Úvodu" je venovaná rôznym možnostiam pre programovanie v jazyku C++, pričom asi najčastejšie čitateľ využije tretiu možnosť – nainštalovať si CodeBlocks v prostredí MS Windows.

Na domácu prácu v jazyku C++ je mnoho možností. Niektoré z nich sú:

1. **Cloud programovanie na on-line kompilátoroch**
2. **Nainštalovať si jednoduché program C4droid na smartfóne alebo tablete s OS Android**
3. **Nainštalovať si CodeBlocks do MS Windows (odporúčame túto možnosť pre MS Windows)**
4. **Nainštalovať si prostredie CodeBlocks do OS LINUX (odporúčame túto možnosť pre Linux)**
5. **Nainštalovať si predinštalovaný virtuálny stroj s OS Linux (Kubuntu)**

Ak ste náročný a chcete si vyskúšať prácu s "veľkým" kompilátorom, môžete si bezplatne nainštalovať Microsoft Visual Studio Express Edition, napríklad z adresy
http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-products

1.1. Cloud programovanie na on-line kompilátoroch

Ak máte prístup na Internet pomocou webového prehliadača, môžete využiť niektorý z on-line kompilátorov. Na napísanie jednoduchého programu sa ani nemusíte registrovať, po zaregistrovaní však získate možnosť ukladať si svoje programy priamo na disk servera, takže v práci môžete pokračovať z ľubovoľného počítača pripojeného na Internet. Do počítača netreba nič inštalovať. Obľúbené cloud-systémy podporujúce bezplatné programovanie v C++ sú:

http://www.sourcelair.com/ - veľmi jednoduché a intuitívne prostredie pre jednoduché programy (nemožnosť zápisu dát programom na disk a ich čítania). Netreba sa ani registrovať. Po zaregistrovaní získate možnosť ukladať svoje programy na serveri, 500 kompilácií/behov programu mesačne je zadarmo.

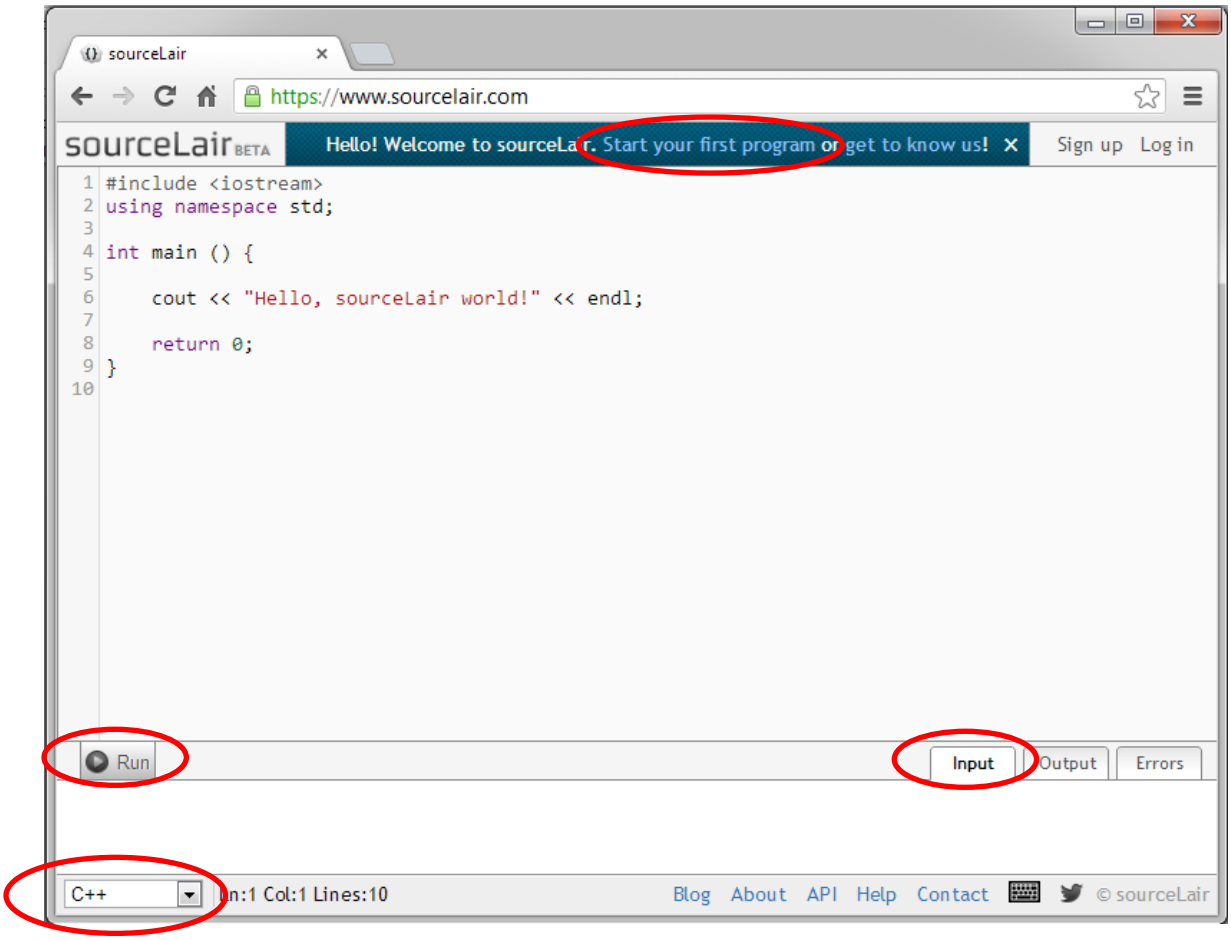
http://ideone.com - podobne ako SourceLair, menej prehľadné prostredie plné reklám, po zaregistrovaní k dispozícii 1000 kompilácií/behov programu mesačne zadarmo.

https://compilr.com/ - veľmi pekné a jednoduché, ale pritom veľmi funkčné prostredie, obsahuje emulátor konzoly (programy bežia rovnako ako na bežnom PC), možnosť programového zápisu a čítania dát zo súborov na serveri, po zaregistrovaní 50 kompilácií/behov programu mesačne zadarmo. Inak sa platí cca 5 EUR mesačne.

Pre ilustráciu ukážeme prácu v systéme SourceLair bez prihlásenia (pozrite obrázok nižšie).

1. Vľavo dolu vyberte jazyk programu: C++
2. Kliknite na linku "Start your first program" hore
3. Upravte predlohu alebo napíšte nový program
4. Stlačte tlačidlo "Run" vľavo dolu

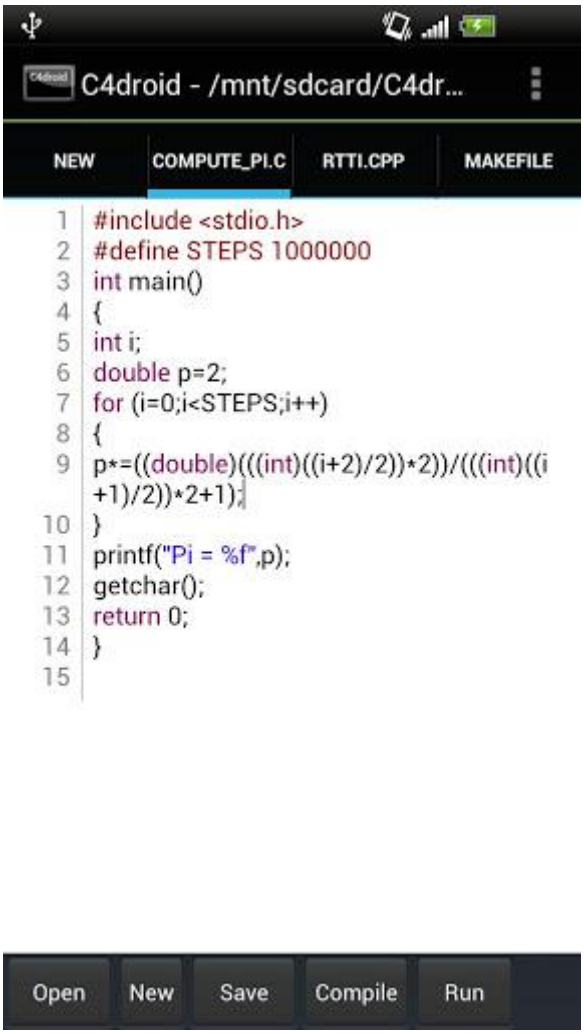
Ak váš program potrebuje vstup z klávesnice, musíte do okna "Input" (jeho záložka je vpravo dolu) pred spustením programu napísať všetko, čo by ste napísali na klávesnici počas behu programu. SourceLair nemá totiž emulátor konzoly, takže nie je možné vkladať údaje počas behu programu. Má to však aj výhodu: ak treba do programu vložiť z klávesnice viac čísel, nemusíte ich pri každom spustení programu vždy písať nanovo, stačí iba upraviť zmeny v nich.

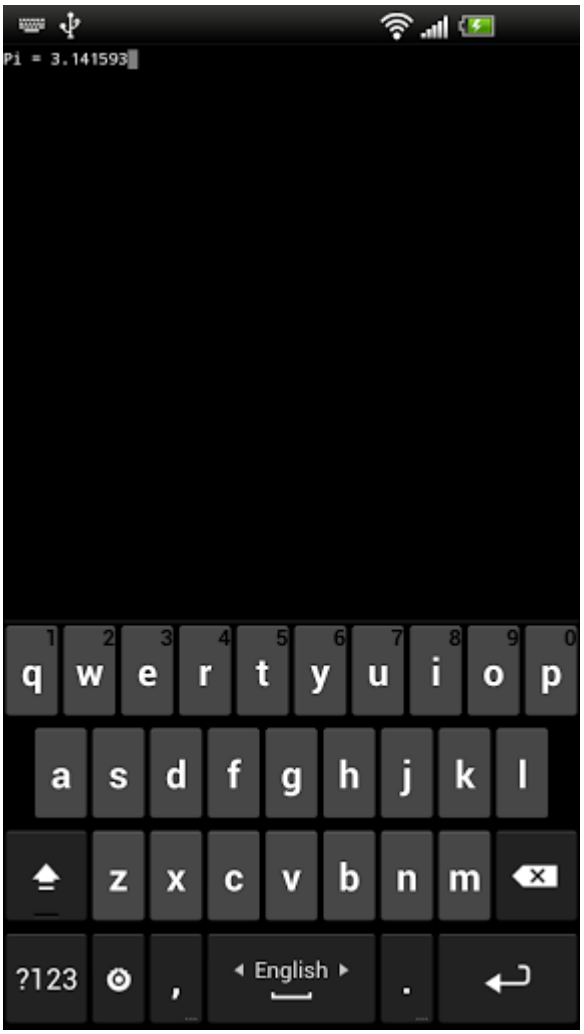


1.2. Ako si nainštalovať jednoduchý program C4droid na smartfóne alebo tablete s OS Android

Ak chcete pracovať naozaj hocikde, nemáte notebook, ale vlastníte smartfón alebo tablet s OS Android, môžete si z GooglePlay nainštalovať jednoduchý systém pre programovanie v C++ s názvom "C4droid". Má isté obmedzenia, ale knižnica <iostream> a od nej odvodené knižnice <fstream> a <stringstream>, ktoré sa najčastejšie používajú pre vstupy z klávesnice, výstupy na obrazovku a pre zápis a čítanie súborov na disku, sú zabudované a fungujú spoľahlivo. Ak by ste chceli, kompilátor možno priamo použiť aj na programovanie aplikácií pre Android (aj grafických - kompilátor podporuje knižnicu Qt). Emulátor konzoly takisto funguje spoľahlivo, takže programy bežia prakticky rovnako, ako na "normálnom" PC. Pre potreby C++ treba doinštalovať z GooglePlay aj kompilátor GCC (ako zásuvný modul). Na doinštalovanie GCC vás vyzve priamo program C4droid.

Program C4droid stojí iba asi 2 eurá a je naozaj dobrý. Plugin GCC je zadarmo. Treba upozorniť, že verzie programu C4driod a pluginu GCC sú spárované. Ak si nainštalujete staršiu verziu programu C4droid, tento nespozná nainštalovanú novšiu verziu pluginu GCC. V súčasnosti je aktuálnou verziou programu C4droid verzia 3.73, inštalačný program má meno "C4droid (CC++ compiler) v3.73.apk".





1.3. Ako si nainštalovať CodeBlocks do OS Windows

Na FMFI UK sa často pracuje v OS Linux. Na písanie programov a kompilovanie sa používa integrované vývojové prostredie (IDE) s názvom CodeBlocks. Všetky uvedené programy sú v licencii GPL (voľne dostupné a šíriteľné ďalej) a sú portované aj pre OS Windows.

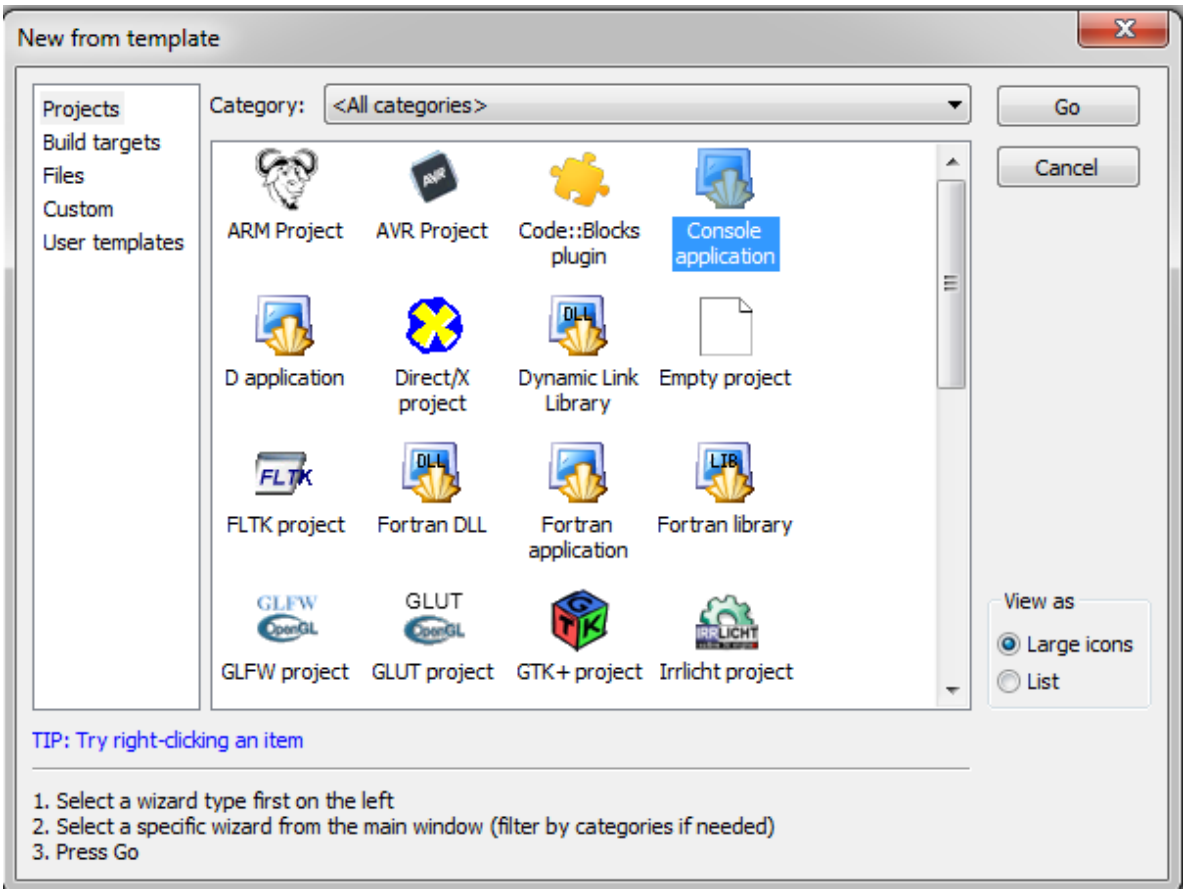
1.3.1. Postup inštalácie CodeBlocks.

IDE CodeBlocks si môžete stiahnuť z adresy http://www.codeblocks.org/downloads/binaries .

Z ponuky inštalačných programov vyberte ten, ktorý už má v sebe zabudovaný aj balík MinGW (obsahuje kompilátor GCC). Momentálne je aktuálnou verziou inštalátora codeblocks-12.11mingw-setup.exe.

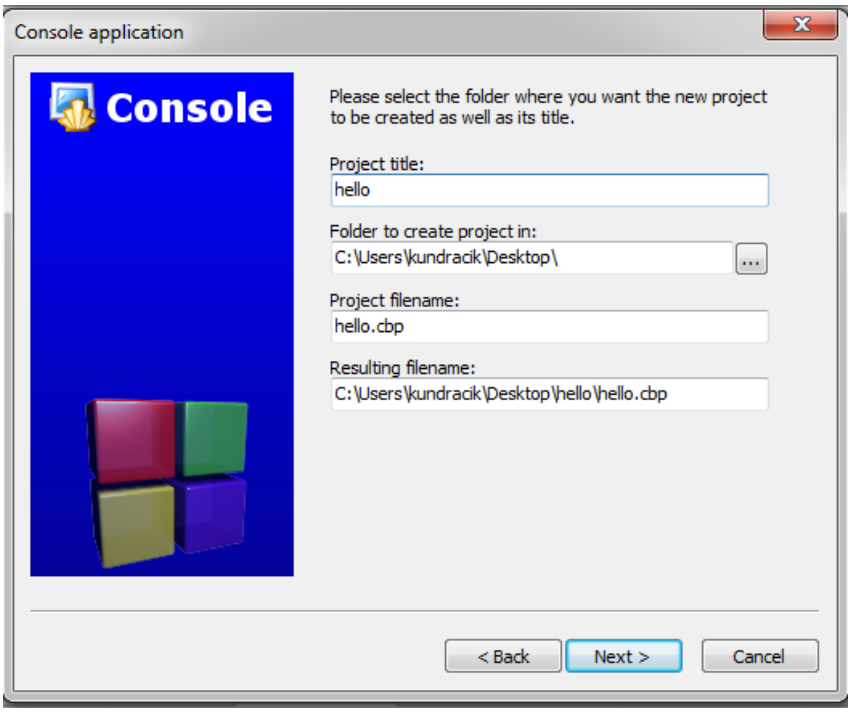
1.3.2. Vytvorenie jednoduchého projektu v CodeBlocks

V menu File/New/Project vyberte Console Application:

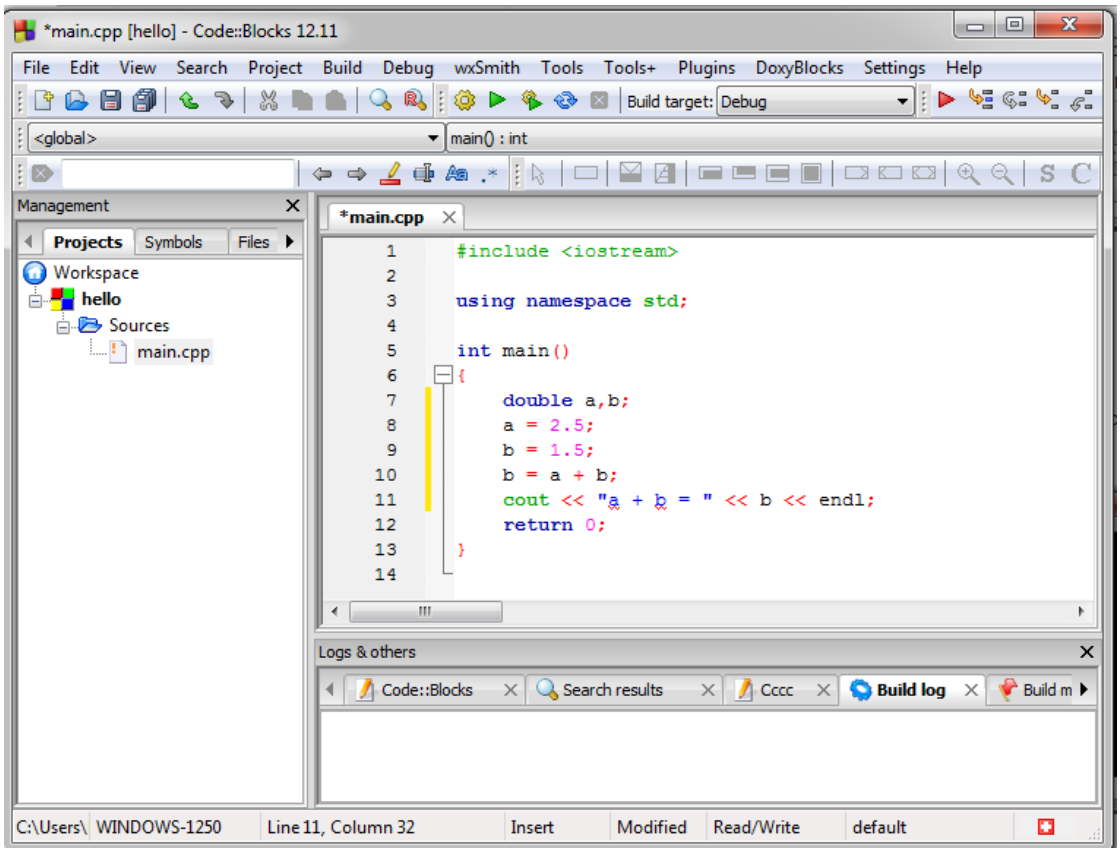


Po stlačení Go (a prípadne ešte Next) vás program vyzve na zvolenie typu programu. Zvoľte C++ a stlačte Next. Potom treba zadať meno projektu (= meno priečinku na uloženie všetkých dát projektu) a adresár, v ktorom sa má adresár s projektom umiestniť (obrázok nižšie). V učebniach odporúčame adresár "net", ktorý sa nachádza na klastri daVinci, takže ho budete mať k dispozícii aj v iných učebniach (alebo doma). Ako meno projektu pre tento pokus dajte "hello".

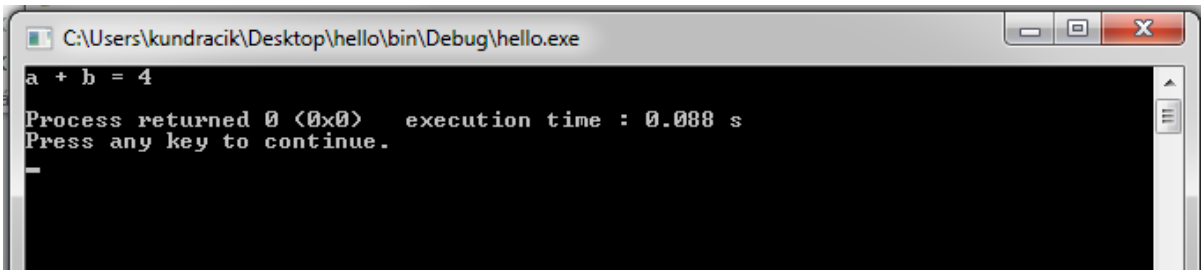
Po stlačení Next máte ešte možnosť zmeniť kompilátor, ktorý sa má použiť. Vyberte (resp. ponechajte bez zmeny) kompilátor GCC, rovnako ponechajte bez zmeny ostatné nastavenia a stlačte Finish.



Upravte text súboru main.cpp nasledovne:



Pomocou menu Build/Build skompilujte program. Ak prebehla kompilácia bez chýb, spustite ho pomocou Build/Run. Spustí sa emulátor konzoly, v ktorej beží program:

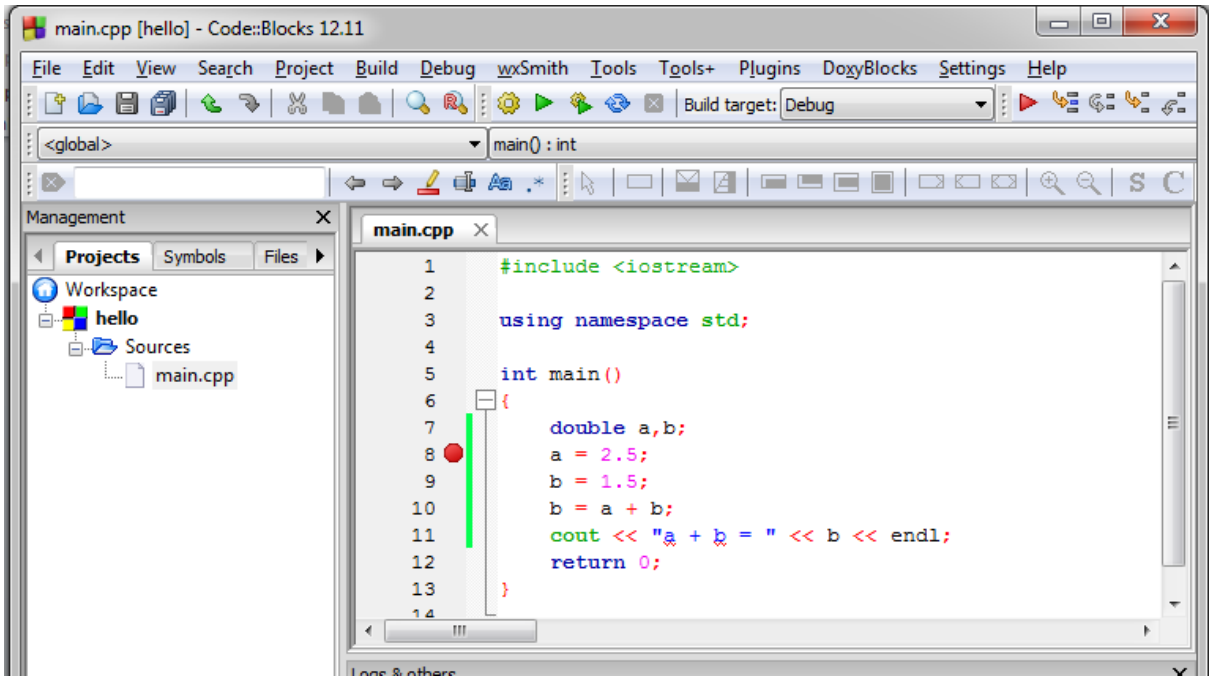


Všimnite si, že po skončení programu ostáva konzola otvorená, čo je veľmi príjemné. V priečinku "hello" pribudli nové súbory. okrem main.cpp (kód programu) pohľadajte v priečinku "bin" skompilovaný program hello.exe.

1.3.3. Ladenie programu v CodeBlocks

V prostredí CodeBlocks možno aj pohodlne ladiť program, t.j. sledovať, po ktorých príkazoch program beží a aký je obsah premenných. Ladiť však možno iba programy skompilované v režime "Debug", ako vidno na obrázkoch vyššie aj nižšie pod hlavným menu. Ďalším módom je "Release", kde sa údaje pre debugovanie nepridávajú, takže výsledný program zaberá na disku menej miesta.

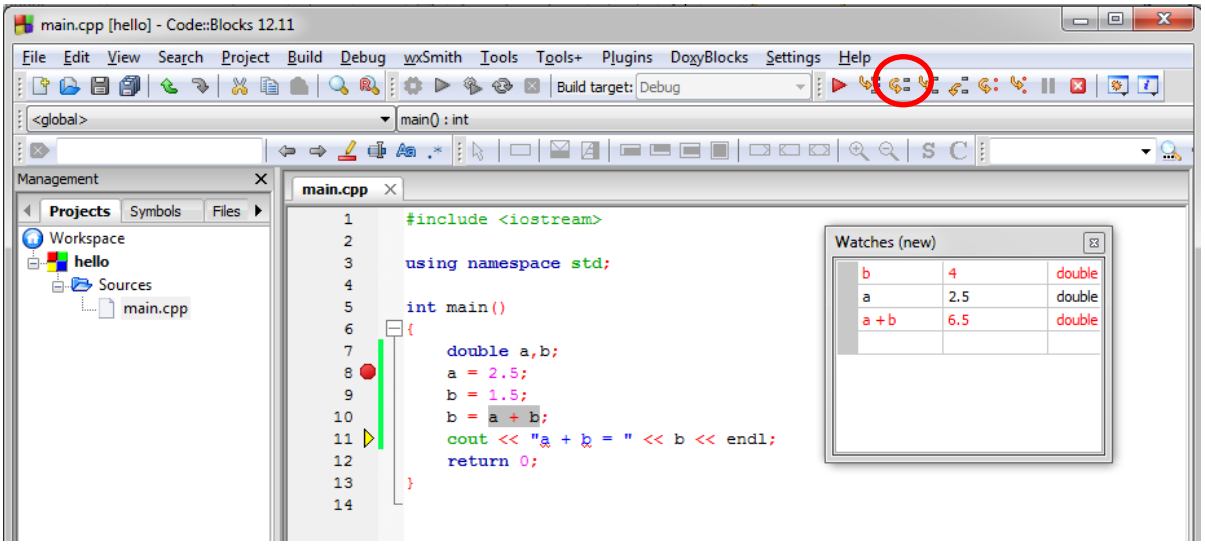
Pred ladením si na vhodné miesto programu (kde ho chceme zastaviť), vložíme tzv. breakpoint. Je to miesto, na ktorom program zastane a jeho ďalší beh budeme môcť ovládať ručne. Breakpoint sa vkladá kliknutím na miesto vpravo od čísla príslušného riadku, napríklad 8:



Riadok s breakpointom sa zvýrazní červeným krúžkom vedľa čísla riadku. Do programu môžete dať aj niekoľko breakpointov na rôzne miesta – beh programu sa na nich zastaví.

Ladenie spustíme pomocou menu Debug/Start. Program sa spustí (objaví sa aj čierne okno konzoly), ale zastane na riadku 8. Riadok, na ktorom program stojí, je označený žltou šípkou. Kliknutím na ikonku "Next line" (vpravo hore pod hlavným menu – na obrázku nižšie je označená krúžkom) sa program postupne vykonáva riadok za riadkom (obrázok nižšie). Počas ladenia môžeme sledovať hodnoty premenných. Treba na ne pridať tzv. "Watch". Robí sa to tak, že kurzor umiestnite na premennú, ktorú chcete sledovať (napríklad "a"), stlačíte pravé tlačidlo myši a z roletkového menu

vyberiete "Watch" (napríklad Watch "a"). Sledovať môžete aj nejaký výraz, napríklad "a+b". Treba označiť myšou výraz, a z roletkového menu vybrať Watch "a+b" . V našom prípade program stojí na riadku 11 (čaká sa na jeho vykonanie), preto má "a" má hodnotu 2.5, "b" má hodnotu 4 a "a+b" má hodnotu 6.5.



Ak chcete, aby program pokračoval ďalej až po najbližší breakpoint (alebo až do konca), stlačte "Debug/Continue" (červená šípka). Program môžete kedykoľvek nasilu ukončiť pomocou "Debug/Stop debugger" (červený štvorček s "x"). Ďalšie triky pri ladení sa naučíte neskôr.

1.4. Ako si nainštalovať CodeBlocks do OS LINUX

Ak vlastníte počítač s OS Linux, všetko potrebné (GCC kompilátor, CodeBlocks) je pripravené v tzv. balíčkoch. Ak ste zvládli nainštalovať si iný potrebný softvér, iste zvládnete aj tieto dva balíčky.

Ukážka práce s nimi je v nasledujúcej podkapitole.

1.5. Ako si nainštalovať predinštalovaný virtuálny stroj s OS Linux (Kubuntu)

Veľmi elegantnou možnosťou, ako pracovať doma v OS Linux so softvérom ako v učebni, je nainštalovať si virtuálny stroj. Program Virtual Box je zadarmo a vo verziách pre všetky bežné operačné systémy (Windows, Linux, OS X, Solaris). Po spustení programu sa vám v okne domovského operačného systému nabootuje nový počítač (napríklad aj s iným operačným systémom). Máte tak možnosť napríklad vo Windows 7 virtuálne prevádzkovať počítač s OS Linux. Obrovskou výhodou je, že takýto virtuálny počítač možno prenášať medzi rôznymi fyzickými počítačmi, alebo ho možno klonovať.

Pripravili sme vám virtuálny stroj s OS Kubuntu 12 s predinštalovaným všetkým potrebným: C/C++ kompilátor GCC, CodeBlocks, Grace, ...

Aké sú požiadavky na váš počítač? Každý aspoň trochu modernejší počítač má dostatočný výkon na prevádzkovanie virtuálneho stroja. Najdôležitejšou požiadavkou je preto **dostatok operačnej pamäte**. Rozumným minimom sú 2 GB RAM, 1 GB pre domovský systém a 1 GB pre virtuálny počítač. Ideálne sú 3 GB RAM alebo viac. Pripravený virtuálny stroj zaberá na harddisku 8GB (takú veľkosť má harddisk virtuálneho stroja) – to by pri dnešných kapacitách harddiskov nemal byť problém.

1.5.1 Nainštalovanie programu Virtual Box a virtuálneho stroja Kubuntu

Postup inštalácie programu Virtual Box závisí od konkrétneho operačného systému. Preto budeme inštaláciu ilustrovať na Window 7 32-bit.

Inštalačný program si stiahnite z https://www.virtualbox.org/wiki/Downloads .

Po nainštalovaní si stiahnite virtuálny stroj Kubuntu z adresy:
ftp://guest:guest@158.195.19.3/guest/ZakladyProgramovania/kubuntu.ova ,
súbor Kubuntu.ova uložte na vhodné miesto vo vašom počítači.

Spustite Virtual Box, pomocou menu File/Import Appliance importujte zo súboru Kubuntu.owa virtuálny stroj.

Spustite virtuálny stroj Ubuntu a môžete pracovať:

Ak si systém bude pýtať meno alebo heslo, tu sú:

user: kubuntu
password: kubuntu

1.5.2 Vytvorenie jednoduchého programu v prostredí CodeBlocks

Práca s prostredím CodeBlocks je rovnaká ako v OS Windows. Preto si pozrite kapitoly 3.2 a 3.3.

1.6. Literatúra o programovacom jazyku C++

1. http://www.cprogramming.com/tutorial/c++-tutorial.html – dobrý on-line tútoriál C++, veľa odkazov na ďalšiu literatúru.

2. http://www.cplusplus.com/files/tutorial.pdf - C++ tútoriál v PDF.
   Na stránke http://www.cplusplus.com je veľa ďalších odkazov na literatúru.

3. http://homel.vsb.cz/~moz017/cpp/ - dobrý on-line tútoriál v češtine

4. http://homel.vsb.cz/~moz017/cpp/kniha/c++.pdf - PDF kniha o C++ pre začiatočníkov v češtine

5. http://www.pragsoft.com/books/CppEssentials.pdf - kniha v PDF, k dispozícii zadarmo aj na Google Play pre mobilné zariadenia s OS Android

2. Asmanov psychrometer, tvoríme jednoduchý program

2.1. Prehľad vedomostí potrebných pre účasť na cvičení

- Do počítača v učebni sa prihlasujete rovnakým menom a heslom, ako do informačného systému AIS2. Na všetkých počítačoch vo všetkých učebniach na fakulte je nainštalovaný rovnaký SW, pracovať teda možno v ľubovoľnej učebni. Vstup do učební je na preukaz študenta/učiteľa, do T3 je vstup zatiaľ na kľúč (je na vrátnici F1).

- Prehľadávanie obsahu diskov pomocou File managera "Konqueror", v "Home" adresári je remapovaný priečinok "net". Tento sa nachádza na klastri daVinci (davinci.fmph.uniba.sk) a je prístupný z ktoréhokoľvek počítača v ktorejkoľvek učebni na fakulte. Dá sa k nemu pristupovať aj z domu pomocou SFTP (vo Windows si nainštalujte program WinSCP). Ostatne priečinky HOME-adresára sú lokálne (na konkrétnom počítači) a pravidelne sa ich obsah vymazáva. Neslúžia teda na odkladanie svojich programov.

- Základná štruktúra programu v C++ je:

```
#include <iostream>
using namespace std;
int main()
{
      cout << "Hello, world!" << endl;
      return 0;
}
```

- Základné dátové typy sú:
  - `int, long` – pre celočíselné premenné so znamienkom (`long` pojme väčšie číslo)
  - `unsigned int, unsigned long` – pre celočíselné premenné bez znamienka
  - `float, double, long double` – pre desatinné čísla
  - `bool` – pre logické hodnoty `true` a `false`

- Premenné možno (ale nie je nutné) deklarovať tak, že môžeme priamo nastaviť ich hodnotu. Platia iba v rámci bloku (časť programu medzi zátvorkami { ... }), v ktorom boli deklarované.
  - `int i;`
  - `int j=0;`
  - `float a=1.0, b=2e3, c=-2.5e-2;`

- Výpis z programu sa robí pomocou `cout`, výpisy možno reťaziť, `endl;` posunie výpis na nový riadok.

```
float a = 2.5;
cout << "a = " << a << endl;
```

- Vstup čísla z klávesnice do premennej sa robí pomocou `cin`:

```
float a;
cin >> a;
```

2.2. Vstup a výstup

Upravte program hello.cpp na sčítavanie dvoch čísel zadaných z klávesnice. Doprogramujte teda vstup z klávesnice do premenných typu float:

```
float a,b,c;
cout << "Zadaj a: ";
cin >> a;
cout << "Zadaj b: ";
cin >> b;
cout << "a + b = " << a+b << endl;
```

Vyskúšajte (doplnením programu) formátovanie výstupu pomocou knižnice `<iomanip>` - nezabudnite ju "inklúdnuť".

`cout << setprecision(3);` - nastaví presnosť 3 platné cifry (nie počet cifier za desatinnou bodkou - to iba v módoch fixed a scientific)

`cout << fixed;` - nastaví mód pevného počtu desatinných miest (v spolupráci so `setprecision()`)

`cout << scientific;` - nastaví mód výpisu vo vedeckej notácii (spolupracuje aj so `setprecision()`)

Príklad výpisu čísla s presnosťou na 2 desatinné miesta:

```
#include <iomanip>
cout << "a + b = " << fixed << setprecision(2) << a+b <<endl;
```

Zastavenie behu programu pred jeho ukončením (inak sa obvykle okno konzoly pri spustení programu v konzole po skončení programu automaticky zavrie a nebudete mať čas pozrieť si výsledky vypísané programom) dosiahneme dvoma príkazmi:

```
cin.ignore(); //načíta a ignoruje všetky znaky, ktoré užívateľ
              prípadne nastláčal na klávesnici

cin.get();    //načíta jeden znak z klávesnice (až po stlačení
              ENTER).
```

V našom prípade sa načítaný znak nepoužije. Netreba ani napísať nejaký znak, stačí stlačiť `ENTER` a program skončí.

2.3. Oboznámenie sa s knižnicou <cmath>

Vyskúšajte si najdôležitejšie funkciíe tejto knižnice (`sin, cos, tan, ln, exp...`). Všimnite si, že hoci sa na umocňovanie `pow(x,n)` používajú logaritmy, <cmath> zvládne aj záporné hodnoty parametra *x*. Uvedomte si rozdiel medzi `atan(x)` a `atan2(y,x)` pri určovaní smeru dvojrozmerného vektora. Hoci `atan(y/x)` obvykle zlyhá pre *x*= 0, knižnica <cmath> zvládne ako argument aj výraz `1/0 = inf`. Príklady vyskúšajte modifikovaním programu.

2.4. Výpočet relatívnej vlhkosti pomocou Asmanovho psychrometra

Relatívnu vlhkosť vzduchu (a iné parametre spojené s vlhkosťou vzduchu) možno určiť meraním teploty suchého a ofukovaného vlhkého teplomera. Z vlhkého teplomera sa odparuje voda, preto bude studenší. Čím suchší vzduch, tým lepšie sa voda odparuje, a tým chladnejší bude vlhký teplomer.

Efekt sa dá predviesť aj s obyčajným izbovým teplomerom, ak máme prístup k jeho zásobníku (guličke), aby sme ju mohli omotať vlhkou vatou. Ideálny je teplomer s presnosťou odčítania zo stupnice asi 0,2 stupňa.

Ak si to chcete vyskúšať, tu je postup:

1. Odmerajte teplotu vzduchu v miestnosti teplomerom, napríklad 22 stupňov
2. Tenký (naozaj tenký) chumáčik vaty navlhčite vodou a obaľte ním "guličku" teplomera
3. Teplomer treba "ovievať" tak, aby rýchlosť vzduchu okolo teplomera bola aspoň 3 m/s (podľa literatúry). Najlepšie sa to robí tak, že teplomer chytíme pevne do ruky a celým ramenom zvoľna (asi 1x za sekundu) krúžime.
4. Počas krúženia občas sledujeme pokles teploty teplomera a počkáme na jej ustálenie (asi 1 až 2 minúty). Odčítame ustálenú hodnotu, napríklad 15 stupňov.

Na výpočet relatívnej vlhkosti slúžia fyzikálne tabuľky. Pre potreby programovania použite analytickými funkciami aproximované tabelované závislosti. Význam premenných vo vzťahoch:

ts – teplota suchého teplomeru
tv – teplota vlhkého teplomeru
dt = *ts*-*tv*

Relatívnu vlhkosť (vzťah je dostatočne presný pre relatívnu vlhkosť nad 10%) vypočítame ako:

```
rv = 100 - dt*(18.47 - 0.8855*ts - 0.1788*dt + 0.02768*ts*ts
     + 0.0004678*dt*dt + 0.001430*ts*dt
     - 0.0003537*ts*ts*ts + 0.00006527*dt*dt*dt);
```

Možno vypočítať aj iné parametre (hustota nasýtených pár pre danú teplotu, absolútna vlhkosť, rosný bod):

mnas – hustota nasýtených pár (v g/m^3)

```
mnas = 4.555+0.3547*ts+0.007351*ts*ts+0.0003685*ts*ts*ts;
```

m – absolútna vlhkosť (v g/m^3)

```
m = mnas*rv/100.0;
```

ros – rosný bod (v ^{o}C)

```
ros = 16.6323/x - 16.6323*x - 6.64947;
```

kde

```
x = pow((a-b)/(a+b),0.16666667);
```

kde

```
a = sqrt(0.431807*m*m - 2.08397*m + 7.47971);
```

```
b = 0.657116*m - 1.58568;
```

2.5. Domáca úloha – zohľadnenie presnosti merania teploty

Úloha: Napíšte program, ktorý z klávesnice načíta dve uvedené teploty *ts* a *tv* a vypočíta relatívnu vlhkosť, absolútnu vlhkosť a rosný bod. Relatívnu vlhkosť vypíšte bez desatinných miest, ostatné údaje na jedno desatinné miesto.

Testovacie dáta: *ts* = 22, *tv* = 15. Výsledok: *rv* = 47 %, *m* = 9,2 g/m^3, *ros* = 10,1 ^{o}C

Doplňte program tak, aby sa z klávesnice zadávala aj presnosť odčítania zo stupnice teplomera, napríklad *er* = 0,2 ^{o}C. Vypočítajte a vypíšte do obrazovky interval možných výsledkov. Uvedomte si, že najnižšia možná vlhkosť zodpovedá teplotám *ts* + *er*, *tv* – *er*. Najvyššiu možnú vlhkosť dostaneme pre *ts* – *er* a *tv* + *er*.

Keďže zatiaľ nevieme nič o volaní funkcií a podprogramov, najjednoduchšie je riadky výpočtu zduplikovať a vyrátať dve sady hodnôt *rv_max*,*rv_min* resp. *m_max*, *m_min*, *ros_max*, *ros_min*.

Testovacie dáta: *ts* = 22, *tv* = 15, *er* = 0,2 Výsledok: *rv* = <44 ; 49>, *m* = <8,9 ; 9,6>, *ros* = <9,4 ; 10,6>

3. Hľadanie koreňa funkcie, iterácie

3.1. Prehľad vedomostí potrebných pre účasť na cvičení

- Na riadenie vykonania / nevykonania časti programu podľa podmienky slúži príkaz `if...else`.
  Má štruktúru:

```
if(podmienka)
{
      prvý blok príkazov;
}
else
{
      druhý blok príkazov;
}
```

  Ak je podmienka splnená, vykoná sa prvý blok príkazov, inak sa vykoná druhý blok príkazov.
  Príklad:

```
if(a == 3)
{
      x = 0;
      y = 2*a;
}
else
{
      x = a;
      y = 1;
}
```

- Časť `else` sa môže vynechať, blokom môže byť aj jediný príkaz (vtedy sa zátvorky pre označenie bloku nemusia použiť). Príklad jednoduchého príkazu `if` pre ohraničenie hodnoty premennej *x*:

```
if(x > 1000) x = 1000;
```

- Porovnávacie príkazy pre podmienky: `<` , `<=` , `==` (rovné) , `>=` , `>` , `!=` (nerovné)

- Skladanie podmienok: `and` (alebo `&&`) , `or` (alebo `||`), `!` (negácia)

- Na opakované vykonanie časti programu ale s rôznou hodnotou nejakého parametra slúži príkaz `for`.
  Štruktúra príkazu:

```
for(vstupné príkazy; podmienkové príkazy; príkazy pre postup
cyklu)
{
      príkazy tela cyklu;
}
```

Pred vstupom do cyklu sa vykonajú vstupné príkazy (oddelené čiarkou). Potom sa otestuje, či je splnená podmienka. Ak nie, príkaz `for` sa ukončí. Ak áno, vykoná sa telo cyklu. Potom vykonajú príkazy pre postup cyklu a otestuje podmienka pre pokračovanie cyklu.
Príklad súčtu čísel 1 až 100, telo cyklu je jediný príkaz (netreba zátvorky označujúce blok príkazov:

```
int sucet =0;
for(i=1; i<=100; i++) sucet += i;
cout << sucet;
```

- `príkaz for možno násilne zmeniť príkazmi:`
  `break;` - príkaz sa ukončí
  `continue;` - ukončí sa beh tela cyklu, pokračuje sa príkazmi pre postup cyklu a testovaním podmienky pre ďalšie vykonanie tela cyklu

- Inými príkazmi cyklu sú `while` a `do...while`

```
while(podmienka)
{
      príkazy tela cyklu;
}
```

resp.

```
do
{
      príkazy tela cyklu;
}
while(podmienka)
```

Telo cyklu sa vykonáva dovtedy, kým je splnená podmienka. V príkaze `do...while` sa telo cyklu vykoná vždy aspoň raz (aj keď podmienka nie je splnená).

- Na opakované použitie časti programu sa používajú funkcie. Ich kód sa píše samostatne (mimo funkcie `main`). Návratová hodnota sa umiestňuje za príkaz `return`. Príklad funkcie, do ktorej vstupujú dve čísla, funkcia vracia ich súčin:

```
float Sucin(float x, float y)
{
      return x*y;
}
```

- Pred prvým použitím funkcie sa v programe musí vyskytnúť alebo jej kód, alebo jej deklarácia:

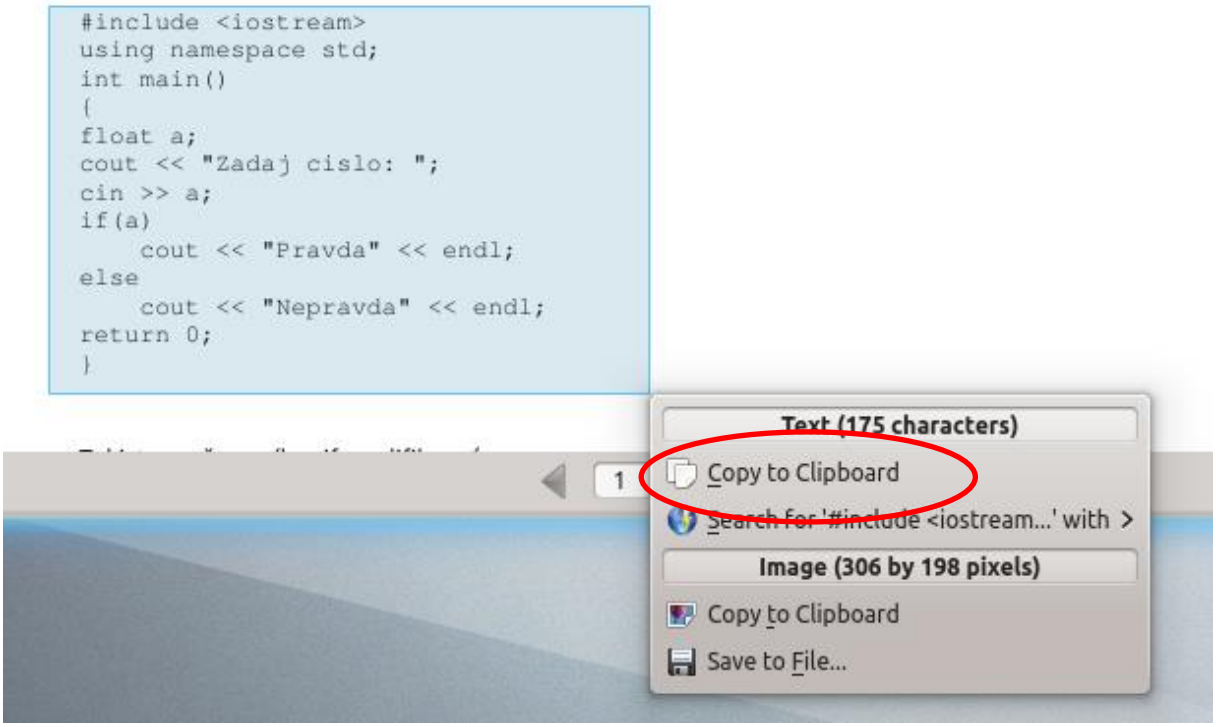
```
#include <iostream>
float Sucin(float,float); //deklaracia funkcie (odporucane)
int main()
{
      float a=2, b=3,c;
      c  = Sucin(a,b);  //vyvolanie funkcie
      cout  <<  c;
}
```

```
float Sucin(float x, float y) //kod funkcie
{
      return x*y;
}
```

- Funkcia nemusí nič vracať, vtedy je návratovým typom `void`.

Pripravené programy v tomto dokumente si môžete rovno skopírovať do CodeBlocks nasledovným postupom:

1. V CodeBlocks zvoľte File/New/Empty file a potom zvoľte File/Save As... a súbor uložte s príponou cpp.
2. Otvorte si www-stránku s týmto dokumentom v prehliadači Konqueror, nie v prehliadači Mozilla Firefox.
3. Označte myšou časť textu s programom a potom zvoľte v roletkovom menu: Text:/Copy to Clipboard(obrázok nižšie). Možno budete najprv musieť zvoliť nástroj pre označovanie (ikonka obrázku orámovaného čiarkovanou čiarou).



4. V CodeBlocks stlačte Ctrl+V.

3.2. Príkaz if-then-else

Jednoduchá ilustrácia príkazu. Každý výraz (aj číselný) má v C++ pravdivostnú hodnotu – možno overiť programom:

```
#include <iostream>
using namespace std;
int main()
{
float a;
cout << "Zadaj cislo: ";
cin >> a;
if(a)
    cout << "Pravda" << endl;
else
    cout << "Nepravda" << endl;
return 0;
}
```

Takisto možno príkaz `if` modifikovať na

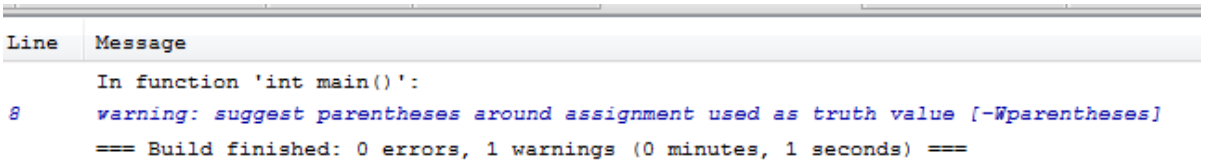
```
if(!a)
```

alebo

```
if(a = 2)
```

V poslednom príklade sa dá ľahko pomýliť. Začiatočníci si priradenie často pomýlia s porovnávaním. Ak do programu vložíme číslo 1 (uloží sa do premennej *a*), v príkaze `if(a = 2)` sa priradí do premennej *a* nová hodnota 2. Výsledkom priradenia je 2, čiže pravda. Začiatočníci sa často pomýlia a myslia si, že výsledkom má byť nepravda, keďže do premennej *a* sme vložili 1, a to nie je rovné 2.

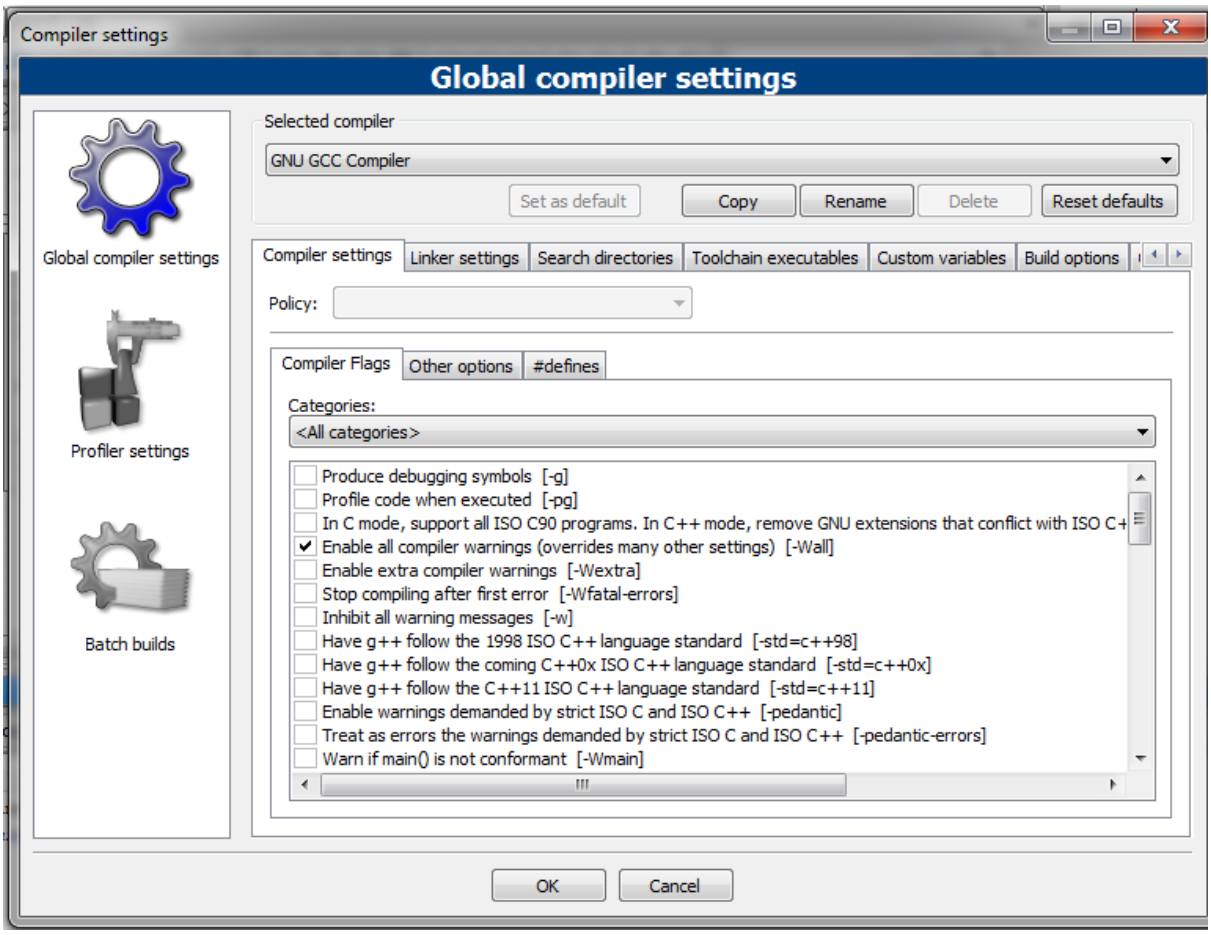
Je to veľmi častá chyba, preto je dobré na začiatok požiadať kompilátor, aby nám dal upozornenie, či to naozaj myslíme ako priradenie, ktorého výsledok sa má použiť ako pravdivostná hodnota. V CodeBlocks menu Settings/Compiler sa vyplatí zaškrtnúť "Enable all compiler warnings" – pozrite obrázok nižšie. Pri kompilácií nás kompilátor upozorní na podozrivé veci. V prípade `if(a=2)` je to upozornenie:



Kompilátor nám odporúča dať priradenie do ďalších zátvoriek, aby to bolo čitateľné a správne aj v prípade komplikovanejších výrazov, napríklad:

```
if((a = 2) == b)
```

V tomto príkaze sa najprv do premennej *a* priradí 2, a potom sa otestuje, či výsledok priradenia (čiže 2) je rovný premennej *b*.

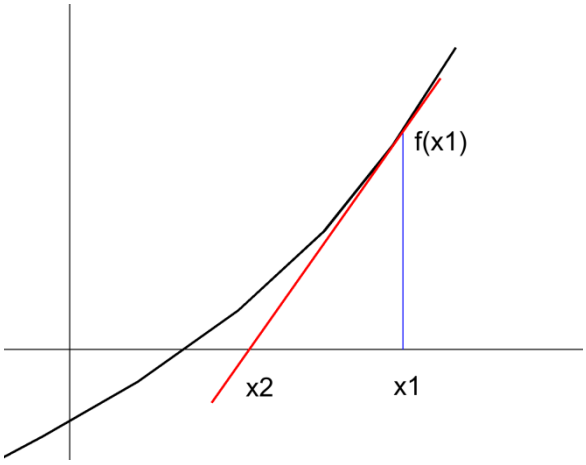


Nastavenie kompilátora na vypisovanie upozornení

3.3. Cyklus while, Newtonova metóda hľadania koreňa funkcie

V dobe bez počítačov bolo postupné polenie intervalov pomerne zdĺhavé a namáhavé. Newton preto prišiel s metódou, ktorá vedie k výsledku veľmi rýchlo.

Veľmi pekne je metóda vysvetlená na stránkach Wikipédie:
http://en.wikipedia.org/wiki/Newton's_method



Princíp je jednoduchý. V bode *x1* (odhad koreňa) nájdeme dotyčnicu k funkcii (na obrázku je červenou farbou). Smernicou dotyčnice je derivácia funkcie v bode *x*1, čiže *f'*(*x*1). Rovnica dotyčnice (priamky) je:

$$y(x) = f'(x1) \,.\, (x - x1) + f(x1)$$

Bod x2 nájdeme ako priesečník dotyčnice s osou *y*, teda:

$$y(x2) = f'(x1) \,.\, (x2 - x1) + f(x1) = 0$$

odkiaľ

$$x2 = x1 - f(x1) / f'(x1)$$

Opakovaním tohto výpočtu získavame stále presnejšie (bližšie ku koreňu) hodnoty *x*2.

Tu je výsledný algoritmus:

1. Odhadneme polohu koreňa *x*1
2. Vypočítame novú, spresnenú hodnotu koreňa: *x*2 = *x*1 – *f*(*x*1)/*f'*(*x*1), kde *f'*(*x*1) je hodnota derivácie funkcie *f*(*x*) v bode *x*1. Hodnotu *x*2 vypíšeme na obrazovku
3. Ak je už rozdiel *x*2-*x*1 malý (napríklad menší než `1e-7`), *x*2 je hľadaným koreňom. Inak za *x*1 dáme hodnotu *x*2 a pokračujeme bodom 2

Upozornenie: veľmi dôležité je nájsť aspoň približnú hodnotu koreňa. Inak metóda nemusí konvergovať a vypočítané hodnoty *x* sa môžu od koreňa aj vzďaľovať.

Úloha: Naprogramujte Newtonovu metódu hľadania koreňa funkcie $x^2 - 2 = 0$, čiže výpočet druhej odmocniny *2*. Na opakovanie krokov 2 a 3 vo vyššie uvedenom algoritme použite cyklus do { ... } while(fabs(x2-x1)>1e-7);

Kontrola: druhá odmocnina z 2 je 1,41421

Všimnite si, že výpočet druhej odmocniny nie je veľmi citlivý na presnosť prvého odhadu, úplne postačí aj hodnota, ktorej druhú odmocninu chceme vypočítať (teda v našom prípade hodnota 2). Ďalej si všimnite, že rýchlosť nájdenia koreňa je vysoká – každým ďalším iteračným krokom sa počet platných desatinných miest výsledku zdvojnásobí.

Úloha: Upravte predchádzajúci program tak, že z klávesnice zadáte číslo, ktorého druhú odmocninu chcete vypočítať. Ako prvý odhad koreňa *x*1 použite priamo hodnotu, ktorú ste zadali z klávesnice.

Kontrola: druhá odmocnina z 3 je 1,73205

Výpočet koreňa funkcie Newtonovou metódou je užitočný a môže sa nám v budúcnosti zísť. Preto tento výpočet spracujeme do podoby funkcie. Takúto funkciu budeme môcť v budúcnosti jednoducho skopírovať do iného programu a ihneď začať používať. Podľa toho, koreň akej funkcie budeme potrebovať hľadať, upravíme vo funkcii iba výpočet výrazu *f*(*x*)/*f'*(*x*).

Úloha: Rozdeľte predchádzajúci program tak, že vo funkcii main() z klávesnice zadáte číslo, ktorého druhú odmocninu chcete vypočítať, a na výpočet druhej odmocniny zavoláte funkciu

double newton(double x). Hodnotu, ktorú vrátila funkcia newton(x), vypíšte do obrazovky. Tým funkcia main() skončí. Samotný výpočet Newtonovou metódou vložte do funkcie newton(x). Argumentom funkcie newton(x) bude číslo, ktorého odmocninu chcete vypočítať. Návratovou hodnotou funkcie newton(x) bude hodnota odmocniny.

3.4. Domáca úloha – korene funkcie tg(x) – x = 0

Pri hľadaní koreňov niektorých funkcií je potrebné poznať dosť presný odhad výsledku, inak hľadanie koreňa zlyhá. Jednou z takýchto funkcií je aj funkcia tg(x) – x.

Úloha: Upravte funkciu newton(x) v programe z cvičení a nájdite prvé štyri kladné korene funkcie tg(x) – x = 0. Nulu nepovažujeme za kladný koreň.

Kontrola: prvým kladným koreňom je 4,49341

Pomôcka: Odporúčame si načrtnúť graf funkcie tg(x) spolu s funkciou x a tak rýchlo nájsť dobré odhady koreňov. Všimnite si, že korene sú od seba vzdialené približne o $\pi = 3{,}14$. Ak nájdete jeden koreň, ľahko odhadnete ďalší. Overte, že ak sa počiatočný odhad významnejšie líši od výsledku, metóda hľadania koreňa zlyhá.

Úloha: Nájdite, v akom rozsahu sa môže meniť odhad prvého koreňa, aby metóda našla správnu hodnotu koreňa: 4,49341

4. Numerické metódy integrovania

4.1. Prehľad vedomostí potrebných pre absolvovanie cvičenia

- Pri delení dvoch celých čísel je zvyšok celočíselný (v zmysle pravidiel C++ pre automatickú konverziu dátových typov):

```
int i=3, j=2;
float k;
k = i/j; //vysledok je 1, nie 1.5!
```

- Ak chceme "nasilu" zmeniť operandy (a tým aj výsledok) napríklad na typ `float`, treba použiť tzv. cast-operátor (názov dátového typu v okrúhlych zátvorkách):

```
int i=3, j=2;
float k;
k = (float)i/(float)j; //vysledok je 1.5
```

Iné príklady:

```
k = (float)i/j; //staci jeden operand float, aby vypocet
                //prebehol ako float
k = 3.0/2.0; //namiesto k = 3/2
```

- Ak chceme zistiť zvyšok po celočíselnom delení, použijeme operátor %:

```
int i=5, j=2, k;
k = i%j; //vysledok je 1, lebo 5/2 je: 2 zvysok 1
```

Čiže výsledok operácie

```
n%2
```

je 0 pre párne *n* a 1 pre nepárne *n*.

- Ak potrebujete, aby funkcia vrátila naraz niekoľko hodnôt, môžete využiť tzv. referencie. Referencia (odkaz) má pri svojej deklarácii pred menom znak '&' .Ak do odkazu priradíme premennú, znamená to, že odkaz bude na túto premennú "ukazovať". V praxi to znamená, že na označenie premennej odteraz môžeme používať aj nové meno (meno odkazu).

```
int i;
int &n=i;
n = 2; //to isté, ako i = 2, t.j. zmeni sa hodnota premennej
         i, nie premennej n (odkazu)
```

Príklad funkcie, ktorá vracia naraz dve hodnoty (t.j. v hlavnom programe funkcia zmení dve premenné:

```
...
void Trigo(float, float &, float &);
...
int main()
{
...
float x=2,y,z;
Trigo(x,y,z); //Do funkcie posielame meno premennych y,z,
                funkcia prijme referencie
```

```
...
}
void Trigo(float x,float &a, float & b)
{
a = sin(x); //zmeni sa premenna x, na  ktoru ukazuje odkaz a,
                dtto y a b
b=sin(x);
}
```

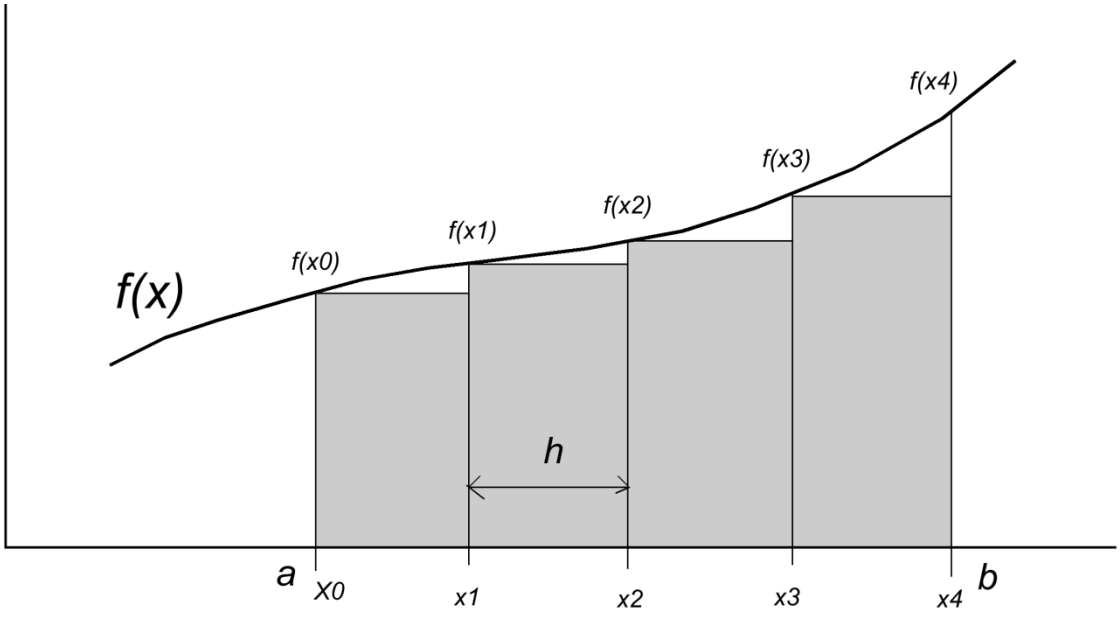
4.2. Obdĺžniková metóda integrovania

Jednorozmerný určitý integrál

$$\int_a^b f(x)dx$$

predstavuje plochu pod ohraničujúcou funkciou *f*(*x*). Rôzne metódy integrovania využívajú rôzne matematické postupy, ako z konečného počtu hodnôt funkcie čo najpresnejšie odhadnúť plochu pod funkciou.

Najjednoduchšou metódou je obdĺžniková metóda (obr.1). Interval <*a*,*b*> sa rozdelí na *n* rovnako širokých intervalov so šírkou *h*. Na obrázku 1 je znázornená situácia pre *n* = 4.



Obr. 1. Obdĺžniková metóda integrovania

Integrál odhadneme súčtom plôch obdĺžnikov (sivá plocha na obrázku):

$$\int_a^b f(x)dx = h \cdot f(x0) + h \cdot f(x1) + h \cdot f(x2) + h \cdot xf(3) =$$

$$= h \cdot \left[f(x0) + f(x1) + f(x2) + f(x3) \right] = h \cdot \sum_{i=0}^{n-1} f(xi)$$

Tento výpočet sa veľmi ľahko programuje. Stačí naprogramovať funkciu *f*(*x*) a vo funkcii `main()` vyzvať užívateľa na vloženie *a*, *b* a *n*. Samotný výpočet je jednoduchý:

```
h = (b-a)/n;
vysledok = 0;
for(i=0;i<n;i++)
{
    vysledok += f(a+i*h);
}
vysledok *= h;
cout << "Vysledok: " << vysledok << endl;
```

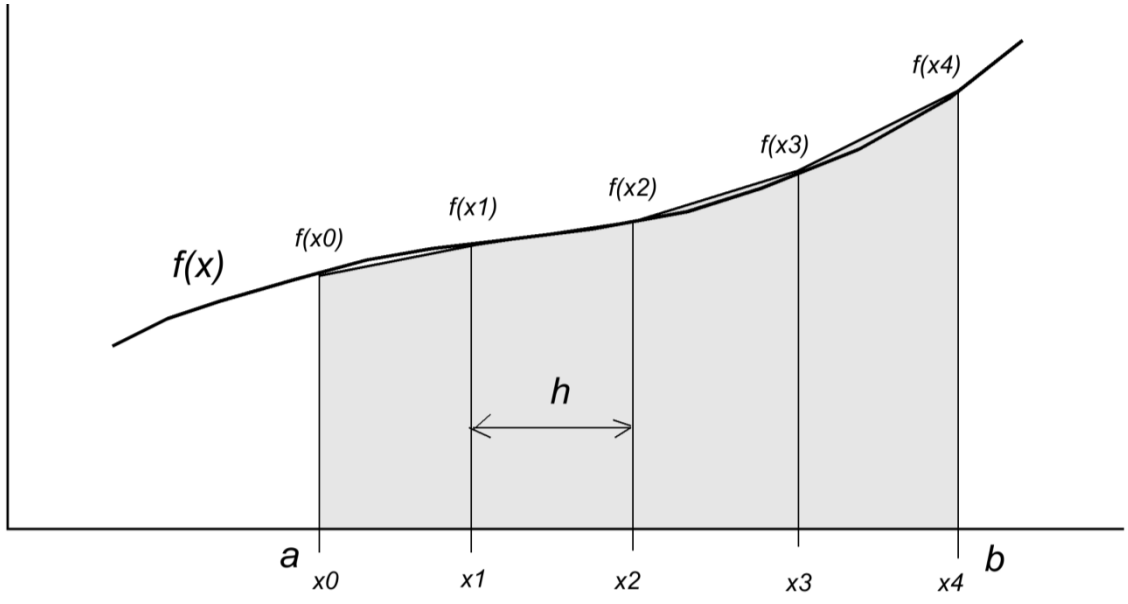
Úloha: Naprogramujte obdĺžnikovú metódu integrovania a vypočítajte ňou

$$\int_0^1 xe^{-x^2} dx = \frac{1-1/e}{2} = 0{,}316060$$

Pri akej hodnote *n* ste dosiahli presnosť 6 desatinných miest?

4.3. Lichobežníková metóda integrovania

Podstatne lepšie výsledky umožňuje získať lichobežníková metóda (obr. 2). Jej podstatou je, že priebeh funkcie medzi dvoma bodmi aproximujeme priamkou.



Obr. 2. Lichobežníková metóda integrovania

Plocha prvého lichobežníka je $h.(f(x0)+f(x1))/2$. Celú plochu získame vzťahom

$$\int_a^b f(x)dx = h\frac{f(x0)+f(x1)}{2} + h\frac{f(x1)+f(x2)}{2} + h\frac{f(x2)+f(x3)}{2} + h\frac{f(x3)+f(x4)}{2} =$$

$$= h\cdot\left[\frac{f(x0)}{2} + f(x1) + f(x2) + f(x3) + \frac{f(x4)}{2}\right] = h\cdot\left[\frac{x0}{2} + \sum_{i=1}^{n-1} f(xi) + \frac{xn}{2}\right]$$

Všimnite si, že vo výsledku sú medzi obdĺžnikovou a lichobežníkovou metódou iba dva rozdiely:

1. Na rozdiel od obdĺžnikovej metódy, v lichobežníkovej metóde sa do výpočtu zahrňuje aj posledný bod (v našom prípade $x4$).

2. Prvý a posledný bod (v našom prípade $x0$ a xn) vstupujú do súčtu polovičnou hodnotou.

Inak je postup výpočtu (a teda aj program) u oboch metód úplne rovnaký.

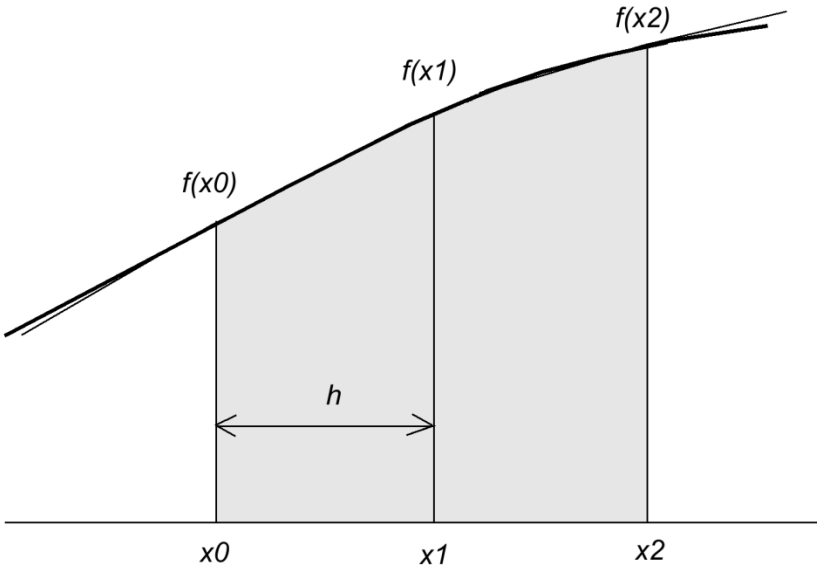
Úloha: Upravte program pre výpočet integrálu obdĺžnikovou metódou tak, aby zahŕňal obe dve zmeny uvedené vyššie. Lichobežníkovou metódou vypočítajte integrál

$$\int_0^1 xe^{-x^2}dx = \frac{1-1/e}{2} = 0{,}316060$$

Pri akej hodnote *n* ste dosiahli presnosť 6 desatinných miest?

4.4. Parabolická (Simpsonova) metóda integrovania

Táto metóda vychádza z aproximovania priebehu funkcie parabolickými úsekmi (obr. 3).



Obr. 3. Aproximácia funkcie parabolou (Simpsonova metóda)

Pre situáciu zobrazenú na obrázku 3 platí pre plochu pod parabolou prechádzajúcou bodmi [*x*0, *f*(*x*0)], [*x*1, f(*x*1)], [*x*2,*f*(*x*2)] vzťah:

$$\int_{x0}^{x2} f(x)dx = \frac{h}{3}\left[f(x0) + 4f(x1) + f(x2)\right]$$

Po zovšeobecnení na *n* bodov pokrývajúcich interval <*a*, *b*>dostávame:

$$\int_{a}^{b} f(x)dx = \frac{h}{3}\left[f(x0) + 4f(x1) + f(x2)\right] + \frac{h}{3}\left[f(x2) + 4f(x3) + f(x4)\right] +$$
$$+ \frac{h}{3}\left[f(x4) + 4f(x5) + f(x6)\right] + \quad \cdots \quad + \frac{h}{3}\left[\cdots + f(xn)\right] =$$
$$= \frac{h}{3}\left[f(x0) + 4f(x1) + 2f(x2) + 4f(x3) + 2f(x4) + \cdots + 4f(x_{n-1}) + f(xn)\right]$$

Ako vidíme, aj v Simpsonovej metóde iba sčítavame funkčné hodnoty v jednotlivých bodoch intervalu. Pre koeficienty, ktorými sa násobia funkčné hodnoty, a pre počet intervalov *n* platí:

1. Počet intervalov *n* musí byť párny.

2. Do súčtu sa zahrňujú všetky body (aj oba konce intervalu integrovania).

3. V prvom (*x*0) a poslednom (*xn*) bode (na hraniciach intervalu integrovania) je koeficient 1.

4. V párnych bodoch (*x*2, *x*4, ...) je koeficient 2.

5. V nepárnych bodoch (*x*1, *x*3, ...) je koeficient 4.

6. Súčet sa vynásobí číslom *h*/3.

Úloha: Upravte program pre výpočet integrálu lichobežníkovou metódou tak, aby zahŕňal zmeny uvedené vyššie. Ošetrite situáciu, ak užívateľ zadal nepárne *n*, tak, že zadanú hodnotu *n* v takomto prípade zväčšíte o 1. Na výpočet zvyšku po delení použite operátor % (príklad: 5%2=1). Simpsonovou metódou vypočítajte integrál

$$\int_{0}^{1} xe^{-x^2} dx = \frac{1-1/e}{2} = 0{,}316060$$

Pri akej hodnote *n* ste dosiahli presnosť 6 desatinných miest?

4.5. Domáca úloha – automatická voľba počtu intervalov

Najväčším problémom pri numerickom integrovaní je vopred určiť, koľko intervalov treba zvoliť, aby sme dosiahli požadovanú presnosť. Jednoduchým (ale nedokonalým) riešením je začať z nejakého počtu intervalov a vypočítať integrál. Potom počet intervalov podstatne (napríklad 2x) zväčšíme a vypočítame integrál znova. Vo zväčšovaní počtu intervalov pokračujeme dovtedy, kým sa novo

vypočítaná hodnota integrálu od predchádzajúcej líši viac než požadovaná presnosť výsledku (napríklad `1e-6`).

Doplňte váš program pre Simpsonovu metódu integrovania vonkajším cyklom, v ktorom postupne budete zdvojnásobovať počet intervalov *n*. Cyklus ukončite, ak sa už dve po sebe vypočítané hodnoty integrálu nelíšia o viac než `1e-6`.

Vaším programom vypočítajte $\int_0^1 x^4 e^{-x^2}\,dx$ s presnosťou na 6 desatinných miest. Aký počet intervalov *n* bol potrebný?

5. Charakteristiky štatistických súborov, práca s poľami

5.1. Prehľad vedomostí potrebných pre absolvovanie cvičenia

- Na prácu s veľkým množstvom údajov sú určené polia. Jednorozmerné pole je analógom vektora, dvojrozmerné pole zodpovedá matici a podobne.

- Deklarácia a použitie jednorozmerného poľa:
```
float x[10]; //deklaracia
x[5]=2; //pouzitie na lavej strane priradenia (ako tzv. lvalue)
x[0]=x[5]; //pouzitie na pravej strane (ako tzv. rvalue)
```
Pole sa indexuje od nuly, t.j. uvedené pole má prvky `x[0]`, `x[1]` až `x[9]`

- Hodnoty prvkov poľa môžeme naplniť už v procese jeho vzniku (pri deklarácii):
```
float x[5] = {2,4,6,8,10};
```

- Rozmer poľa vie kompilátor zistiť aj automaticky zo zadaných hodnôt:
```
float x[]={1,2,3}; //vytvori sa pole s troma prvkami
```

- S viacrozmernými poľami sa pracuje analogicky:
```
float x[10][10]; //deklaracia dvojrozmerneho pola
x[5][1]=2;
x[0][0]=x[5][1];
float y[][]={{1,2,3},{4,5,6},{7,8,9}; //pole 3x3 prvky
```

- V pamäti počítača sú prvky poľa uložené za sebou. U viacrozmerných polí sa najrýchlejšie mení posledný index.

- Programovací jazyk C/C++ nekontroluje hranice polí. Priradením hodnoty do neexistujúceho prvku môžete poškodiť (pozmeniť) obsah iných premenných!

5.2. Aritmetický priemer ako odhad najpravdepodobnejšej hodnoty

Pri spracovaní výsledkov meraní, ktoré sú zaťažené náhodnými vplyvmi, treba z viacerých nameraných hodnôt odhadnúť ich strednú hodnotu a takisto štandardnú odchýlku (neistotu) tohto odhadu.

Strednú hodnotu meranej veličiny najčastejšie odhadujeme aritmetickým priemerom z n nameraných hodnôt x:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Použitie aritmetického priemeru na určenie najpravdepodobnejšej hodnoty meranej veličiny je založené na predpoklade, že namerané hodnoty sú s rovnakou pravdepodobnosťou rozptýlené nad aj pod najpravdepodobnejšou hodnotou.

Štandardnú odchýlku aritmetického priemeru od najpravdepodobnejšej hodnoty určíme vzťahom:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n(n-1)}}$$

Úloha: Naprogramujte výpočet aritmetického priemeru a jeho štandardnej odchýlky od najpravdepodobnejšej hodnoty. Vstupné hodnoty zadajte pri deklarácii premennej *x*:

```
float x[100]={2.6, 3.4, 2.8, 3.8, 3.1, 2.9, 3.6, 2.7, 3.3, 2.5, 3.1, 2.9, 2.8};
int n=13;
```

Kontrola: aritmetický priemer: 3,038, štandardná odchýlka: 0,109.

5.3. Medián ako vhodnejší odhad najpravdepodobnejšej hodnoty pri vybočujúcich údajoch

Pokiaľ sú vo výsledkoch údaje vymykajúce sa bežným hodnotám jedným smerom, nie je aritmetický priemer vhodnou charakteristikou. Typické je uvádzanie priemernej mzdy, ktorá je veľmi zdeformovaná kvôli malému počtu vysokých manažérskych platov. Pri meraní môže podobná situácia vzniknúť napríklad pri chybnom zaznamenaní jedného z údajov. Pri ručnom spracovaní výsledkov si takúto chybu obvykle všimneme, ale pri automatickom spracovaní môže takáto hodnota "prekĺznuť" a úplne znehodnotiť aritmetický priemer.

Medián sa počíta nasledujúcim spôsobom:

1. Prvky (hodnoty) x_i usporiadame podľa veľkosti
2. Ak je počet hodnôt *n* nepárne číslo, mediánom je prostredný prvok. Inak je mediánom aritmetický priemer z dvoch "prostredných" prvkov

5.3.1 Utriedenie čísel

Na utriedenie sa používajú rôzne sofistikované postupy. Pre naše účely však úplne postačí jednoduchý algoritmus "bubble sort", ktorého slovenským prekladom by mohlo byť "prebublinkovávanie".

Postup triedenia metódou "prebublinkovávania" je nasledovný:

1. Prechádzame zaradom cez všetky prvky zoznamu.
2. Ak je daný prvok zoznamu menší, než predchádzajúci, prvky navzájom vymeníme.
3. Takto postupujeme až do konca zoznamu.
4. Ak došlo aspoň k jednej výmene, pokračujeme bodom 1.

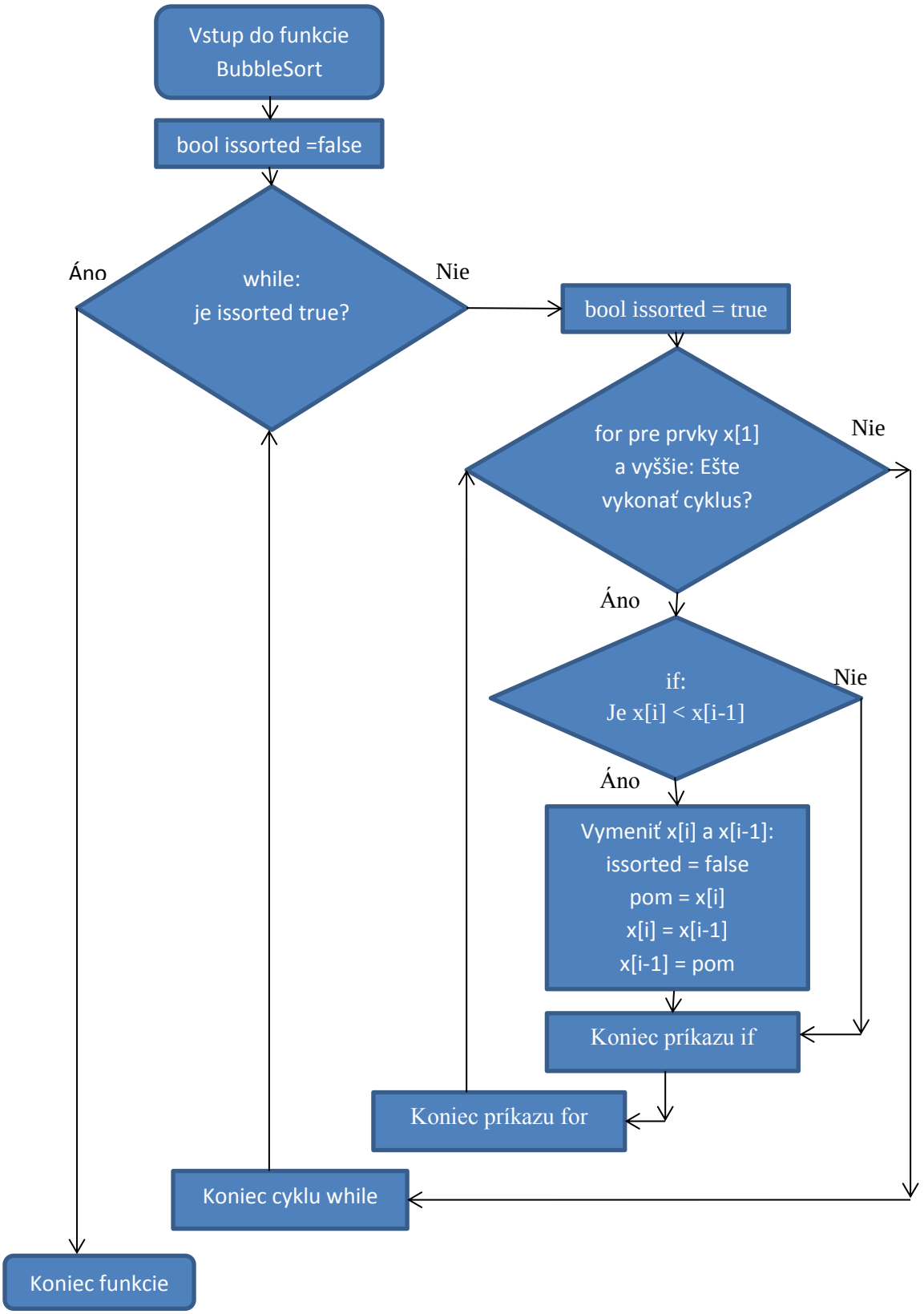
Úloha: Doprogramujte do vášho programu na výpočet aritmetického priemeru a jeho chyby metódu `BubbleSort` podľa vzoru nižšie. Všimnite si, že do funkcie `BubbleSort` vchádza *odkaz* na pole *x*. Funkcia preto utriedi priamo pole *x* deklarované vo funkcii `main()`.

```
void BubbleSort(float (&x)[100], int n)
{
SEM NAPISTE VAS KOD
}
```

Vo funkcii `main()` utrieďte prvky poľa *x* a vypíšte ich do obrazovky príkazmi

```
BubbleSort(x,n);

for(int i=0;i<n;i++)
   cout << x[i] << " ";
cout << endl;
```

Námet: Vo funkcii `BubbleSort` si zadeklarujte premennú typu `bool` (napríklad `bool issorted`), ktorá bude indikovať stav utriedenia. Pred každým prehľadávaním poľa *x* (napríklad cyklom `for(int i=1;i<n;i++))` si ju najprv nastavte na `true`. Ak došlo v cykle k výmene prvkov, nastavte ju na `false`. Pole prehľadávajte od začiatku do konca opakovane dovtedy, kým pole nie je utriedené (na opakovanie môžete využiť napríklad cyklus `while(!issorted))`. Algoritmus funkcie `BubbleSort` je znázornený na nasledujúcom vývojovom diagrame:



Poznámka: Názov pre túto metódu utrieďovania ("bubble sort") sa ujal preto, lebo malé čísla sa nahor dostanú postupom, ktorý veľmi pripomína stúpanie bubliniek vo vode. Aj tam si vzduch (malé číslo) vymení svoju pozíciu s vodou (veľké číslo). Nakoniec všetka voda klesne dolu a všetky bublinky sa dostanú nahor nad hladinu vody.

5.3.2. Výpočet mediánu

Úloha: Ak vám už funguje triedenie, vypočítajte medián (nezabudnite na rôzny spôsob jeho výpočtu pri párnom a nepárnom *n*). Na zistenie parity *n* použite zvyšok po celočíselnom delení dvoma: `n%2`.

Pokiaľ sa medián veľmi nelíši od aritmetického priemeru, v dátach asi neboli údaje vybočujúce jedným smerom.

Kontrola: medián = 2,9

5.4. Domáca úloha – interaktívne vkladanie údajov

Upravte váš program tak, aby sa namerané údaje nezadávali priamo do kódu pri deklarácii premennej *x*, ale z klávesnice. Užívateľ zadá najprv počet nameraných hodnôt a potom si program postupne vypýta samotné namerané hodnoty (využiť môžete napríklad cyklus `for`).

6. Sústava lineárnych rovníc, maticové operácie

6.1. Prehľad vedomostí potrebných pre absolvovanie cvičenia

- Pri používaní funkcií, ktoré pracujú s rozsiahlymi poľami, je výhodnejšie pracovať s referenciami na polia. Ak by sme do funkcie poslali priamo pole, pri každom volaní funkcie sa najprv urobí lokálna kópia poľa pre potreby funkcie. Výsledkom je, že to bude trvať dlho a spotrebuje sa veľa pamäte počítača.

- Príklad použitia referencie na pole (funkcia priamo manipuluje s poľom deklarovanom v hlavnom programe:

```
...
void Vymen(float &x[10][10],int i, int j, int k, int l);
...
int main()
{
float x[10][10];
...
Vymen(x,1,1,2,2); //vymeni navzajom prvky x[1][1] a x[2][2]
...
}
void Vymen(float &x[10][10],int i, int j, int k, int l)
//vstupom je referencia na pole
{
float tmp=x[i][j];
x[i][j]=x[k][l];
x[k][l]=tmp;
}
```

6.2. Oboznámenie sa s funkciou na inverziu matice a výpočet determinantu

Množstvo algoritmov vo fyzike je založených na inverzii matice alebo výpočte determinantu. Jedným z najefektívnejších numerických algoritmov pre inverziu matice je Gaussova-Jordanova eliminačná metóda. Oboznámili ste sa s ňou na prednáške a cvičeniach z algebry. Pri numerickej implementácii tohto algoritmu si treba dať pozor na dve veci:

1. Ak prvok v riadku, ktorý chceme pričítať, je nulový, nie je možné pomocou neho eliminovať prvok v inom riadku. Vtedy treba riadky vhodne vymeniť.
2. Pri sčítavaní veľkých čísel s malými často dochádza k strate presnosti. Preto je vhodné na elimináciu použiť vždy najväčší prvok (tzv. pivot).

Pripravili sme vám funkciu, ktorá pomocou Gaussovej-Jordanovej eliminácie vypočíta inverznú maticu. Zároveň jej návratovou hodnotou je determinant matice (determinant je jednoduchým súčinom

pivotných prvkov – je to vlastne "odpad" pri inverzii matice). V skutočnosti je výpočet determinantu Gaussovou-Jordanovou eliminációu jedným z najefektívnejších.

Vo funkcii `main` programu (pozri nižšie) je ilustrované použitie tejto funkcie. Pri použití si uvedomte, že:

1. Funkcia prijíma referencie na pôvodnú aj inverznú maticu. Pracuje teda priamo s maticami deklarovanými vo vyvolávajúcej funkcii.
2. Vstupná matica (tá čo sa má invertovať) sa priamo používa na elimináciu. Po skončení funkcie je preto jedničkovou maticou (jej obsah sa zničí). Ak pôvodnú maticu budete ešte potrebovať, vyrobte si vo vyvolávajúcej funkcii (v našom príklade je to `main`) jej kópiu a na inverziu použite túto kópiu.
3. Keďže funkcia umiestni výsledok inverzie priamo do matice deklarovanej vo volajúcej funkcii, návratovú hodnotu môžeme využiť ľubovoľne. My sme ju využili na návrat hodnoty determinantu matice.
4. Ak je determinant matice (t.j. návratová hodnota funkcie) nulový, matica je neinvertibilná. Podľa toho, v akej fáze došlo k zlyhaniu výpočtu, bude vyzerať aj obsah pôvodnej a invertovanej matice. Rozhodne však nebude správny. Preto pred ďalším pokračovaním vášho programu otestujte návratovú hodnotu na nulu.
5. Všimnite si, že vo funkcii sa predpokladá maximálny rozmer matice 30x30. Ak potrebujete pracovať s väčšími maticami, zmeňte záhlavie a deklaráciu funkcie `MatrixInversion`. Vo funkcii `main()` deklarujte matice takisto s upraveným maximálnym rozmerom.

Úloha: Prezrite si kód funkcie `MatrixInversion` (nemusíte ho ale pochopiť, skôr skúste identifikovať, na čo slúžia jeho časti).
Dôkladne sa oboznámte s použitím funkcie `MatrixInversion` vo funkcii `main`. Vyskúšajte zadať pri deklarácii matice niektoré jednoduché matice (napríklad 2x2), ktorých inverznú maticu dokážete sami rýchlo vypočítať. Overte správnosť inverzie matice a výpočtu determinantu.

Kód ukážkového programu a funkcie `MatrixInversion`:

```
#include <iostream>
#include <cmath>

using namespace std;
double MatrixInversion(double (&)[30][30], int n, double (&)[30][30]);

int main()
{
    double a[30][30]={{0,1,0},
                      {0,1,1},
                      {1,0,1}};
    int n=3;

    double ainv[30][30];
    double determinant;

    //Vypis povodnej matice
    cout << "Povodna matica: " << endl;
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
```

```
            cout << a[i][j] << "\t";
        cout << endl;
    }
    cout << endl;

    if((determinant=MatrixInversion(a,n,ainv))==0)
        cout << "Matica je neinvertovatelna!"<<endl<<endl;

    //vypis determinantu
    cout << "Determinant = " << determinant << endl << endl;
    //vypis inverznej matice
    cout << "Inverzna matica: " << endl;
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
            cout << ainv[i][j] << "\t";
        cout << endl;
    }

    cout << endl;

    //vypis eliminovanej matice
    cout << "Eliminovana (jednotkova) matica:" << endl;
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
            cout << a[i][j] << "\t";
        cout << endl;
    }

}


double MatrixInversion(double (&A)[30][30], int n, double
(&AInverse)[30][30])
{
    // A = vstupna matica (n x n)
    // n = rozmer matice A
    // AInverse = inverzna matica (n x n)
    // Tato funkcia invertuje maticu Gauss-Jordanovou eliminaciou.
    // Vracia determinant matice. Ak vrati 0, matica sa neda invertovat.
    // Pozor!!! Povodna matica A sa eliminaciou stane jednotkovou!
    int i, j, iPass, imx, icol, irow;
    double det, temp, pivot, factor;
    det = 1;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            AInverse[i][j] = 0;
        }
        AInverse[i][i] = 1;
    }
    // Aktualmnym pivotom riadku je iPass.
    // Pre kazdy prechod - najprv najst najvacsi element v stlpci pivota.
    for (iPass = 0; iPass < n; iPass++)
    {
        imx = iPass;
        for (irow = iPass; irow < n; irow++)
        {
            if (fabs(A[irow][iPass]) > fabs(A[imx][iPass])) imx = irow;
```

```
        }
        // Vymenit prvky riadku iPass a riadku imx v A aj AInverse.
        // Zmenit znamienko determinantu na opacne pri zamene riadkov
        if (imx != iPass)
        {
            det *= -1;
            for (icol = 0; icol < n; icol++)
            {
                temp = AInverse[iPass][icol];
                AInverse[iPass][icol] = AInverse[imx][icol];
                AInverse[imx][icol] = temp;
                if (icol >= iPass)
                {
                    temp = A[iPass][icol];
                    A[iPass][icol] = A[imx][icol]; A[imx][icol] = temp;
                }
            }
        }
        // Aktualnym pivotom je teraz A[iPass][iPass].
        // Determinant je sucinom pivotov.
        pivot = A[iPass][iPass];
        det = det * pivot;
        if (det == 0)
        {
            return 0;
        }
        for (icol = 0; icol < n; icol++)
        {
            // Normalizovat riadok pivotu delenim hodnotou pivotu.
            AInverse[iPass][icol] = AInverse[iPass][icol] / pivot;
            if (icol >= iPass) A[iPass][icol] = A[iPass][icol] / pivot;
        }
        for (irow = 0; irow < n; irow++)
        // Pricitat nasobok riadku pivotu ku kazdemu riadku.
        // Nasobiaci koef. sa zvoli tak, ze prvok A v stlpci pivota je 0.
        {
            if (irow != iPass) factor = A[irow][iPass];
            for (icol = 0; icol < n; icol++)
            {
                if (irow != iPass)
                {
                    AInverse[irow][icol] -= factor * AInverse[iPass][icol];
                    A[irow][icol] -= factor * A[iPass][icol];
                }
            }
        }
    }
    return det;
}
```

6.3. Riešenie sústavy lineárnych algebraických rovníc pomocou inverzie

Sústavu lineárnych rovníc

$$ax_O + bx_1 + cx_2 = d$$
$$ex_0 + fx_1 + gx_2 = h$$
$$ix_0 + jx_1 + kx_2 = l$$

môžeme zapísať v maticovom tvare

$$\begin{pmatrix} a & b & c \\ e & f & g \\ i & j & k \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} d \\ h \\ l \end{pmatrix} \qquad \mathbf{A} \cdot \mathbf{x} = \mathbf{y}$$

Riešením takejto sústavy rovníc je

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{y} \qquad \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} a & b & c \\ e & f & g \\ i & j & k \end{pmatrix}^{-1} \cdot \begin{pmatrix} d \\ h \\ l \end{pmatrix}$$

Príklad sústavy dvoch rovníc:

$$3x_O + 4x_1 = 11$$
$$x_0 + 2x_1 = 5$$

Maticový zápis:

$$\begin{pmatrix} 3 & 4 \\ 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 11 \\ 5 \end{pmatrix} \qquad \mathbf{A} = \begin{pmatrix} 3 & 4 \\ 1 & 2 \end{pmatrix} \qquad \mathbf{A}^{-1} = \begin{pmatrix} 1 & -2 \\ -\frac{1}{2} & \frac{3}{2} \end{pmatrix}$$

Riešenie:

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 1 & -2 \\ -\frac{1}{2} & \frac{3}{2} \end{pmatrix} \cdot \begin{pmatrix} 11 \\ 5 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Úloha: Upravte kód funkcie `main()` tak, aby slúžil na výpočet sústavy rovníc. Obsluha a činnosť programu by mali vyzerať nasledovne:

1. Užívateľ zadá z klávesnice počet rovníc na riešenie *n*
2. Na základe vloženého čísla sa v cykle načítajú rovnice. Každá rovnica sa zadá na jeden krát ako súbor čísel oddelených medzerami alebo tabulátormi, napríklad prvú rovnicu $3x_0 + 4x_1 = 11$ zadáme takto:

```
Zadaj rovnicu 1: 3 4 11
```

3. Podľa počtu rovníc načítame v ďalšom vnorenom cykle prvky jedného riadku matice *a*. Nakoniec ešte načítame prvok vektora *y*. Príklad zadávania a načítania koeficientov rovníc:

```
for(int i=0;i<n;i++) //cyklus po riadkoch (rovniciach)
{
      //Tu vypísať výzvu na zadanie koeficientov rovnice
      for(int j=0;j<n;j++)
            //Tu načítať z cin prvok a[i][j]
      //Tu načítať z cin prvok y[i]
}
```

4. Vypočíta sa inverzná matica
5. Inverzná matica sa vynásobí s vektorom pravých strán, výsledok sa uloží do vektora riešení
6. Vypíše sa riešenie sústavy rovníc

Návod: Detailný vývojový diagram programu je na nasledujúcom obrázku nižšie.

6.4. Domáca úloha – riešenie sústavy rovníc determinantami

Riešenie sústavy rovníc je možné aj pomocou determinantov. Z numerického hľadiska to nie je veľmi efektívna metóda, keďže najrýchlejší výpočet determinantu je beztak spojený s elimináciou matice. Napriek tomu vyskúšame aj tento postup.

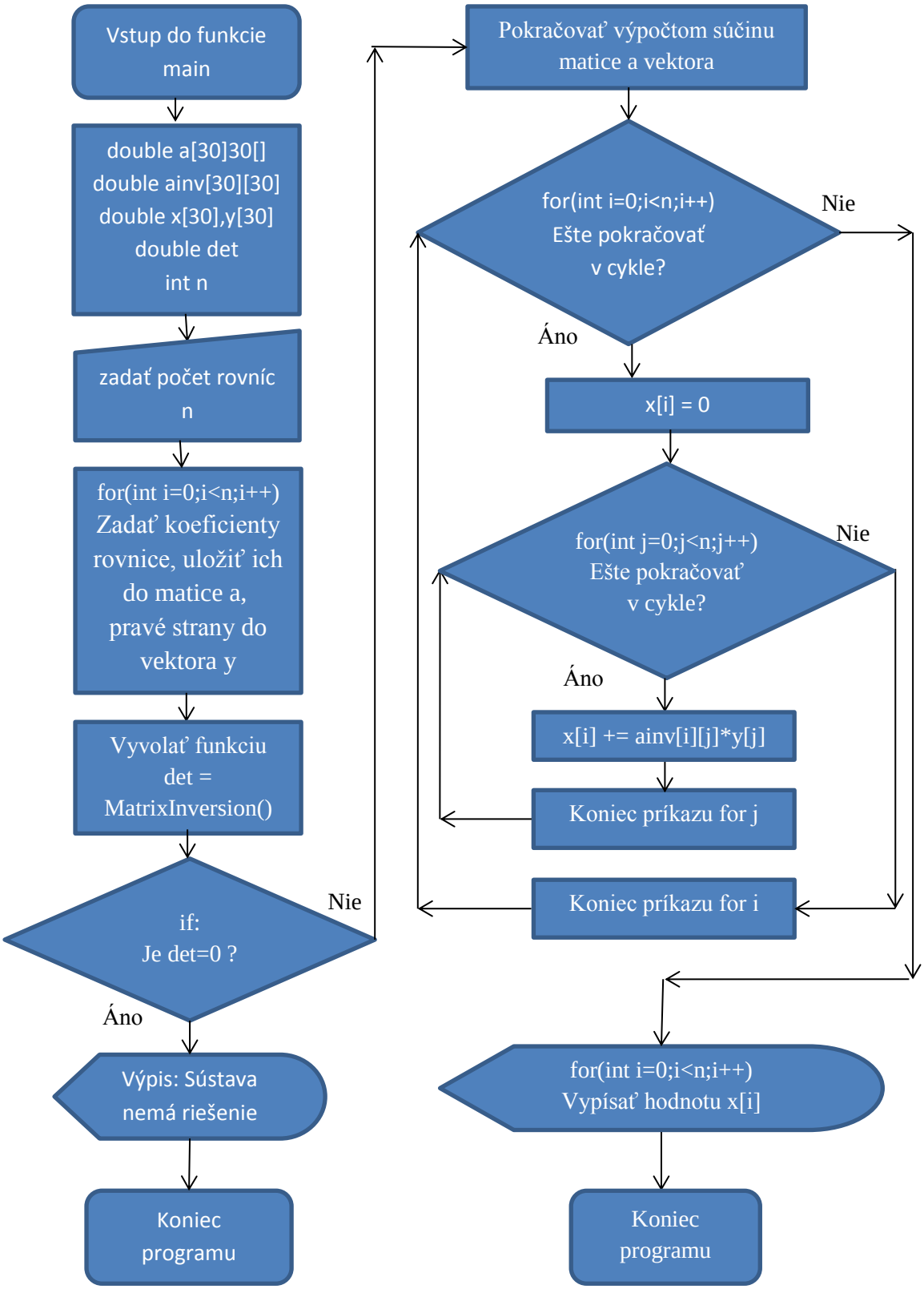
Úloha: Upravte program z cvičení tak, aby vypočítal hodnotu x_0 pomocou podielu dvoch determinantov. V menovateli podielu má byť determinant sústavy (matice *a*), v čitateli má byť determinant matice *a*, v ktorej sme prvý stĺpec (s indexom 0) nahradili pravou stranou sústavy rovníc.

Sústava troch rovníc na riešenie:

$$3x_O + 4x_1 + 2x_2 = 1$$
$$2x_0 + 2x_1 - 3x_2 = 8$$
$$4x_0 - 3x_1 - 7x_2 = 3$$

Potrebné zmeny v programe:

1. V programe si deklarujte aj novú maticu `double b[30][30]` a premennú `double detb`.
2. Za načítaním koeficientov sústavy rovníc v programe vložte dvojitý cyklus, v ktorom prvok za prvkom prekopírujete *a*[*i*][*j*] do *b*[*i*][*j*]. Potom prvý stĺpec (s indexom 0) matice *b* v cykle nahraďte pravou stranou sústavy rovníc (vektorom *y*), t.j. *b*[*i*][0] = *y*[*i*]
3. Vypočítajte determinant sústavy príkazom:
   `det = MatrixInversion(a,3,ainv);`
4. Samotnú inverznú maticu ainv nepotrebujeme, vypočítajte determinant premennej príkazom:
   `detb = MatrixInversion(b,3,ainv);`
5. Vypočítajte `x[0] = detb/det;` a vypíšte hodnotu.


Vstup do funkcie main
double a[30]30[]
double ainv[30][30]
double x[30],y[30]
double det
int n
zadať počet rovníc n
for(int i=0;i<n;i++)
Zadať koeficienty rovnice, uložiť ich do matice a, pravé strany do vektora y
Vyvolať funkciu det = MatrixInversion()
if:
Je det=0 ?
Nie
Áno
Výpis: Sústava nemá riešenie
Koniec programu
Pokračovať výpočtom súčinu matice a vektora
for(int i=0;i<n;i++)
Ešte pokračovať v cykle?
Nie
Áno
x[i] = 0
for(int j=0;j<n;j++)
Ešte pokračovať v cykle?
Nie
Áno
x[i] += ainv[i][j]*y[j]
Koniec príkazu for j
Koniec príkazu for i
for(int i=0;i<n;i++)
Vypísať hodnotu x[i]
Koniec programu

7. Pád telesa vo viskóznom prostredí. Eulerova metóda riešenia diferenciálnych rovníc . Kreslenie grafu funkcie.

7.1. Prehľad vedomostí potrebných na absolvovanie cvičenia

- Pre prácu s dátovými súbormi na disku slúžia toky dát (streamy). Funkcie pre streamy z/do textových súborov sa nachádzajú v knižnici `<fstream>`.
  Na otvorenie súboru pre čítanie slúži stream typu `ifstream`:

```
#include <fstream>
...
ifstream fin("text.txt");
                //otvorenie suboru text.txt na citanie
                //fin je meno streamu (moze byt aj ine meno)
...
```

- Treba vždy otestovať, či sa stream podarilo otvoriť. Ak ste napríklad zadali meno neexistujúceho súboru, stream sa nevytvoril a treba to ošetriť:

```
ifstream fin("text.txt");
if(fin==0) //rovna sa nule pri chybe
     {
     cout << "Nepodarilo sa otvorit subor text.txt");
     return;
     }
```

- Ďalej sa so streamom pracuje rovnako, ako so streamom z klávesnice: všetko, čo je v súbore, ako keby ste napísali na klávesnici, napríklad načítanie čísla napísaného v súbore do premennej *x*:

```
fin >> x;
```

- Vytvorený stream `fin` je vlastne objektom triedy `ifstream`. Dotkli sme sa teda objektového konceptu C++. Objekt v sebe zahŕňa viaceré premenné a pozná taktiež funkcie (metódy) pre manipuláciu s premennými. Pre vyvolanie funkcie sa používa kombinácia mena objektu a funkcie, ktoré sú spojené bodkou, napr. `fin.eof()`.

- Ak chcete zistiť, či ste už načítali celý súbor (čí ste už na jeho konci), použite funkciu:
  `fin.eof()`
  ktorá vráti `true`, ak ste už na konci súboru.

- Po skončení práce so súborom stream (súbor) uzavrieme príkazom:

```
fin.close();
```

- Na otvorenie súboru pre zápis slúži stream typu `ofstream`:
```
#include <fstream>
...
ofstream fout("text.txt");
                //otvorenie suboru text.txt na zapis
                //fout je meno streamu (moze byt aj ine meno)
...
```
Ak súbor "text.txt" ešte neexistuje, automaticky sa vytvorí prázdny súbor s takýmto menom. Ak súbor sa takýmto menom existuje, tak sa otvorí pre zápis *a jeho obsah sa vymaže*.

- Treba vždy otestovať, či sa stream podarilo otvoriť. Ak sa napríklad pokúsite vytvoriť alebo zapisovať do jestvujúceho súboru na CD ROM, určite to nepôjde:
```
ofstream fin("text.txt");
if(fout==0) //rovna sa nule pri chybe
...
```

- Ďalej sa so streamom pracuje rovnako, ako so streamom na obrazovku: všetko, čo by sa napísalo na obrazovku, sa napíše do súboru::
```
fout << "Ahoj!" << endl;
```

- Po skončení práce so súborom stream (súbor) uzavrieme príkazom:
```
fout.close();
```

- Ak nechcete obsah jestvujúceho súboru pre zápisom doň najprv vymazať, ale nové texty chcete pridať na jeho koniec, na otvorenie použite príkaz:
```
ofstream fout("text.txt", ios::app);
```

- Príslušnosť k nejakej triede (opäť sa dotýkame objektového konceptu) sa vyjadruje kombináciou mena triedy a "štvorbodky" pred menom premennej (ale napríklad aj funkcie), napr. `ios::app`. V podstate by sme takýto spôsob mali používať aj pri používaní objektov `cin` a `cout`, kde ich plné meno je `std::cin` a `std::cout`. Aby sme to nemuseli robiť, príkazom `using namespace std;` kompilátoru povieme, že má za nás automaticky pridať príslušnosť `std::` (ak nie je v programe explicitne uvedená iná príslušnosť).

- Z vášho programu môžete spustiť ľubovoľný iný program pomocou funkcie `system()`, ktorá sa nachádza v knižnici `<cstdlib>`
Príklad spustenia programu Notepad (štandardná súčasť MS Windows):
```
#include <cstdlib>
...
system("notepad.exe");
...
```

- Mnohé programy prijímajú ako ďalší parameter pri spustení meno súboru, ktorý majú zobraziť. Ak chceme spustiť Notepad s otvoreným súborom text.txt, urobíme to príkazom:
```
system("notepad.exe text.txt");
```

- Užívateľ však môže na otváranie textových súborov radšej používať iný program, napríklad Wordpad. Užívateľ si spravidla s textovým súborom asocioval jeho obľúbený program, takže po dvojkliku myšou na ikonu textového súboru sa tento automaticky otvorí v obľúbenom programe. Rovnakú akciu môžeme dosiahnuť príkazom `start`:
  ```
  system("start text.txt");
  ```
  Súbor sa automaticky otvorí v obľúbenom (asociovanom) programe.

- Príkaz `start` umožňuje aj to, že užívateľov program počká, kým užívateľ neskončí prácu v spustenom externom programe:
  ```
  system("start /wait text.txt");
  ```

7.2. Vytvorenie dátového súboru na disku, graf funkcie

Vykresľovanie rôznych funkčných závislostí patrí ku každodennej práci fyzika. Na ilustráciu si zobrazíme graf funkcie

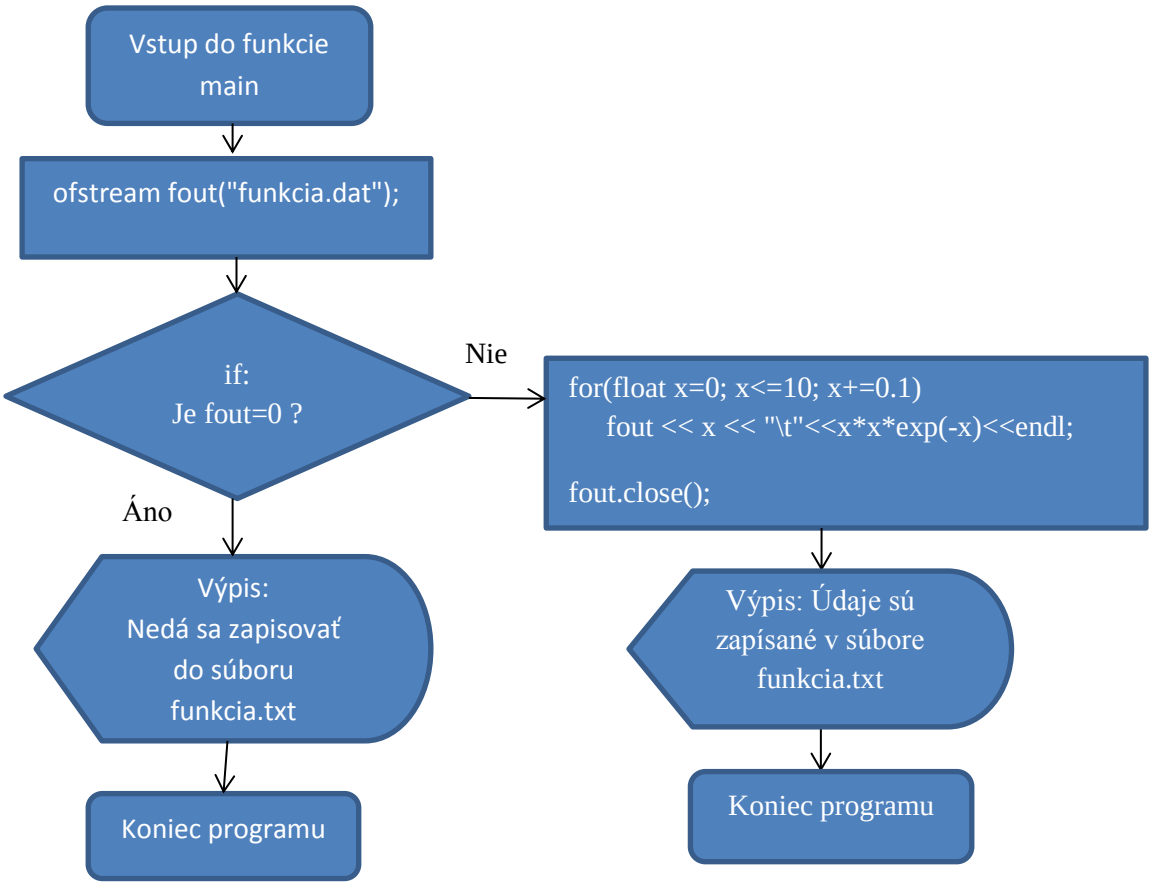
$$y(x) = x^2 e^{-x}$$

Úloha: Napíšte program, ktorý do súboru na disku s názvom "funkcia.dat" uloží body vyššie uvedenej funkcie v rozsahu *x* = <0; 10>.

Návod: Otvorte si stream pre zápis do súboru "funkcia.dat", nech sa napríklad volá `fout`. Overte, že sa stream podarilo otvoriť. Ak nie, program ukončite. Potom do súboru zapíšte dvojice *x*, *y* oddelené tabulátorom, každá dvojica na novom riadku:

```
for(float x = 0; x<=10;x+=0.1)
   fout << x << "\t" << x*x*exp(-x) << endl;
```

Nezabudnite do programu zahrnúť všetky potrebné knižnice. Vývojový diagram:

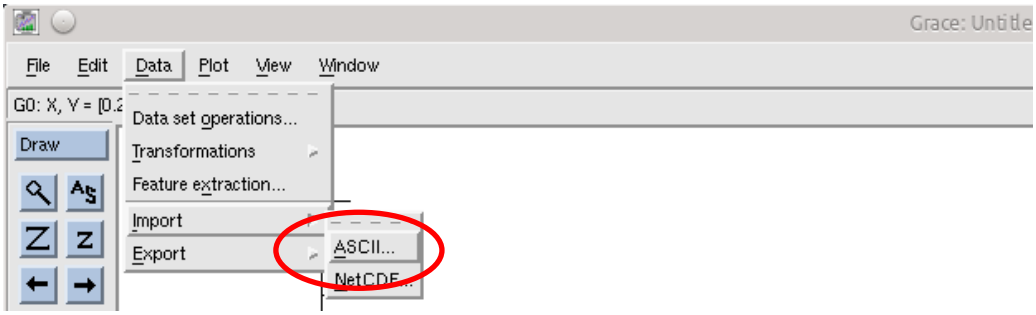


Pozrite si obsah vytvoreného súboru "funkcia.dat" a skontrolujte, že naozaj na jednotlivých riadkoch sú vždy dve čísla oddelené tabulátorom.

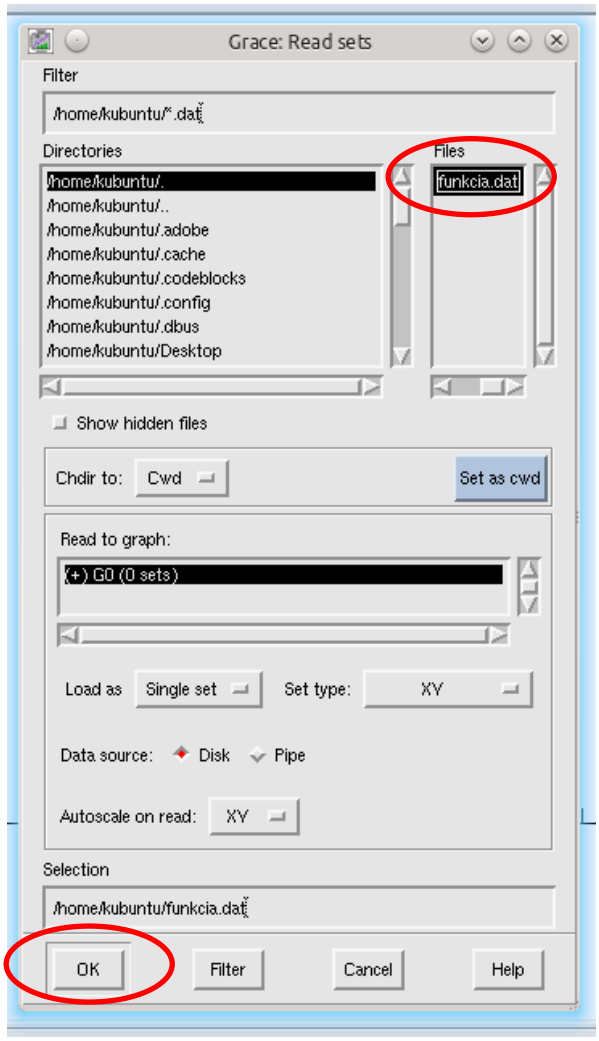
7.3. Zobrazenie dátového súboru (graf funkcie) v programe Grace

Program Grace slúži na zobrazovanie rôznych 2D (t.j. *x*-*y*) grafov v prostredí OS Linux. Návod na prácu nájdete na http://plasma-gate.weizmann.ac.il/Grace/doc/Tutorial.html

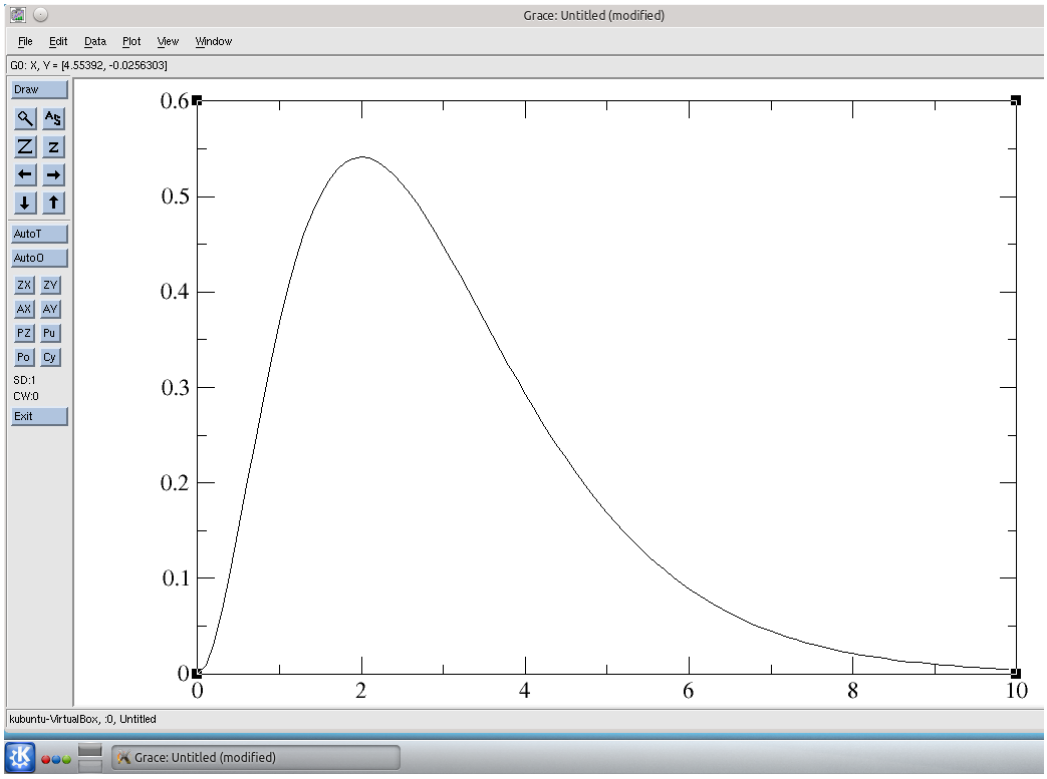
1. Spustite program Grace. Mal by byť umiestnený v skupine "Education/Mathematics".
2. Zvoľte menu "Data/Import/ASCII":



3. Pohľadajte a vyberte súbor "funkcia dat" a stlačte OK:



4. Potom už zavrite dialóg pre import súborov, napríklad stlačením "Cancel". Výsledkom je nakreslený graf:



Ako sme už spomenuli, vykresľovanie vypočítaných údajov je veľmi častou úlohou. Aby ste nemuseli zakaždým ručne načítať vypočítaný súbor, program Grace možno spustiť aj z príkazového riadku, pričom parametrom je súbor na zobrazenie:

```
xmgrace funkcia.dat
```

Program Grace pritom dokážeme spustiť aj priamo z nášho programu vyvolaním funkcie `system()`:

```
system("xmgrace funkcia.dat")
```

Úloha: Doplňte do vášho programu príkaz, ktorý pred skončením programu najprv spustí nakreslenie grafu v Grace z dátového súboru "funkcia.dat".

V prostredí MS Windows si môžete nainštalovať iný program na kreslenie grafov, napríklad Gnuplot. Dobre však poslúžia aj tabuľkové procesory, napríklad MS Excel.

Pre zobrazenie jednoduchého čiarového grafu pomocou Gnuplot vo Windows možno použiť príkaz

```
system("wgnuplot -persist -e \"plot 'funkcia.dat' with lines\"");
```

Program Gnuplot (jediný súbor s menom "wgnuplot.exe") si môžete stiahnuť z adresy:

ftp://guest:guest@158.195.19.3/guest/ZakladyProgramovania/wgnuplot.exe

Nakopírujte si ho do adresára, kde váš program vytvára súbor "funkcia.dat", a môžete ho používať.

7.4. Eulerova metóda riešenia diferenciálnych rovníc s počiatočnou podmienkou

Tento typ úloh je typický napríklad pri výpočte pohybu telies, kde zrýchlenie telesa je určené všetkými silami pôsobiacimi na neho. Na začiatku výpočtu poznáme počiatočnú polohu telesa a jeho rýchlosť, úlohou je vypočítať rýchlosť a prejdenú dráhu telesa v závislosti od uplynutého času pohybu.

Princíp Eulerovej metódy si ilustrujeme na riešení jednoduchej diferenciálnej rovnice opisujúcej pád guľôčky vo veľmi viskóznom prostredí (predpokladáme laminárne prúdenie a zanedbávame vztlakovú silu):

$$m\frac{dv}{dt} = mg - 6\pi r \eta v$$

kde r je polomer guľôčky, m je jej hmotnosť g je tiažové zrýchlenie, η je dynamický koeficient viskozity kvapaliny a v je rýchlosť guľôčky.

Uvažujme oceľovú (hustota 7800 kg/m^3) guľôčku s polomerom 2 mm padajúcou v hustom oleji (koeficient dynamickej viskozity 1 Pa.s). Vtedy po vyčíslení konštánt dostávame diferenciálnu rovnicu v tvare

$$\frac{dv}{dt} = 9{,}81 - 144v$$

Poznámka: ak chcete aplikovať aj vztlakovú silu, treba za hustotu guľôčky vziať rozdiel hustôt materiálu guľôčky a kvapaliny. Napríklad pre oceľovú guľôčku (7800 kg/m^3) a olej (600 kg/m^3) treba použiť hustotu 7200 kg/m^3.

Eulerova myšlienka je nahradiť spojitý priebeh času t výpočtom rýchlosti iba v diskrétnych bodoch t_i, ktoré sú od seba vzdialené v pravidelných (veľmi krátkych) intervaloch Δt. Vtedy možno deriváciu (a celú diferenciálnu rovnicu) v bode t_i približne vyjadriť v tvare:

$$\frac{dv}{dt} \approx \frac{v_i - v_{i-1}}{\Delta t} = 9.81 - 144 v_{i-1}$$

Takúto rovnicu (kde veličiny sú iba diskrétne a deriváciu nahradíme výpočtom rozdielov) nazývame *diferenčnou*. Riešenie tejto diferenčnej rovnice je jednoduché:

$$v_i = v_{i-1} + (9{,}81 - 144 v_{i-1})\Delta t$$

Čiže, ak poznáme hodnotu rýchlosti v_{i-1} v čase t_{i-1}, výpočet rýchlosti v_i v čase t_i je triviálny. Celý proces výpočtu sa naštartuje na začiatku pohybu, kedy je rýchlosť guľôčky známa, napríklad $v_0 = 0$ m/s. Potom už ľahko vypočítame pomocou uvedeného vzťahu rýchlosť v_1 v čase Δt, ďalej rýchlosť v_2 v čase $2\Delta t$, a tak ďalej...

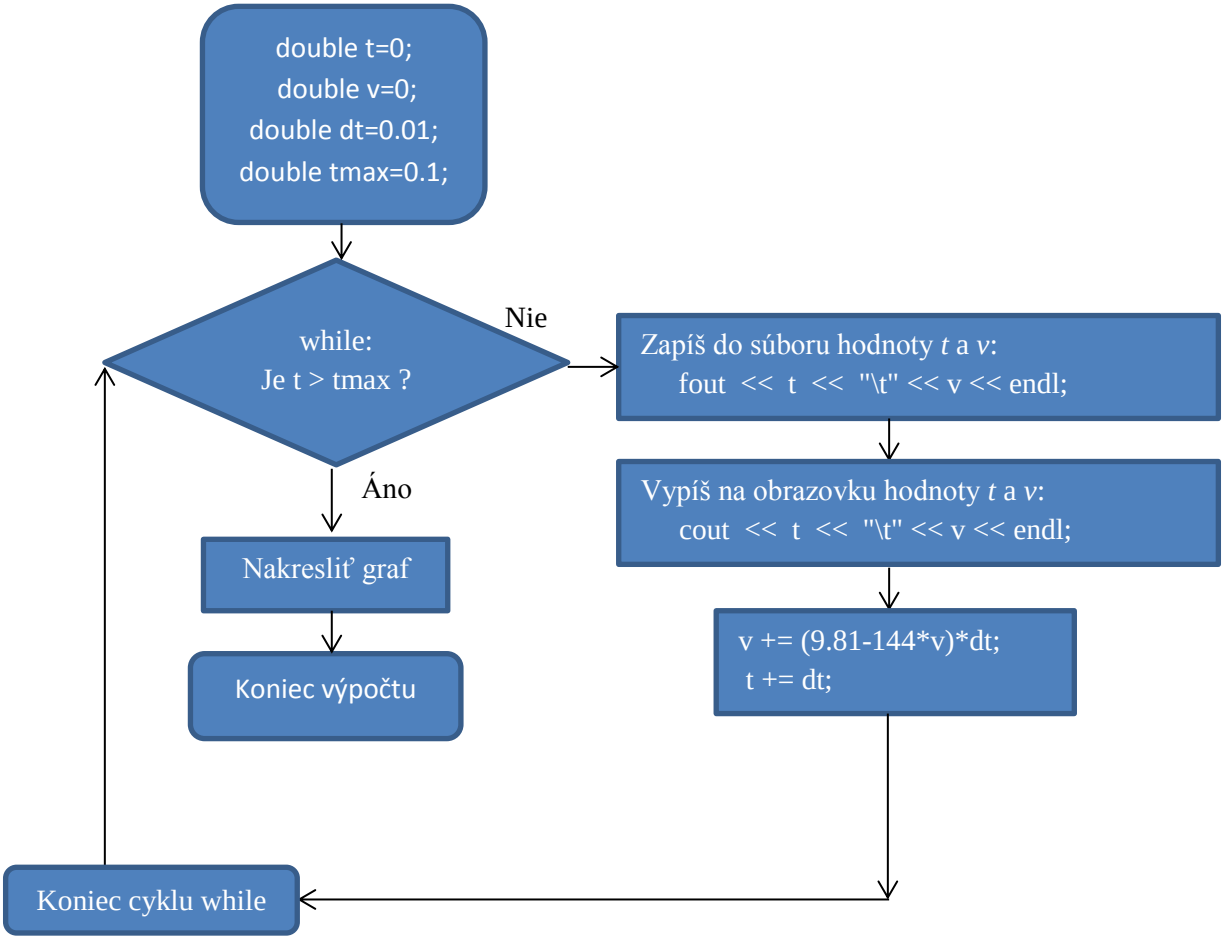
Úloha: Upravte váš program tak, aby počítal Eulerovou metódou pohyb guľôčky vo viskóznom prostredí. Na začiatku pohybu predpokladajte nulovú rýchlosť guľôčky. Do súboru "rychlost.dat" ukladajte dvojice vypočítaných hodnôt t_i a v_i oddelené tabulátorom, každá dvojica na novom riadku. Program skončite po dosiahnutí doby pohybu 0,1 s.

Aký krok musíte zvoliť, aby po jeho zjemnení nebolo vidieť podstatné zvýšenie presnosti? Za ako dlho sa rýchlosť guľôčky ustáli? Aká je ustálená hodnota rýchlosti? Pozor! Celkovú dobu

pohybu a časový krok voľte uvážene, aby ste nepočítali hodnoty v miliónoch bodov! Jednak to bude dlho trvať, jednak bude problém tieto hodnoty zobraziť vo forme grafu! Začnite s krokom 0,01 s.

Čo sa stane, ak dáte krok priveľmi hrubý (napríklad 0,1 s) a necháte počítať 1 s pádu? Pri akej dĺžke kroku dokáže váš program vypočítať zmysluplné výsledky počas 1 s pádu guľôčky?

Návod. Vývojový diagram výpočtového jadra programu je na nasledujúcom obrázku.



7.5. Domáca úloha – skok z hranice vesmíru

V nedávnej dobe v médiách rezonoval rekord v skoku na Zem z hranice vesmíru (z výšky 39 km). V takomto prípade je rýchlosť pádu limitovaná aerodynamickým odporom za prítomnosti turbulentného prúdenia. Vtedy odporová trecia sila závisí od druhej mocniny rýchlosti v telesa:

$$F_t = \frac{1}{2} C S \rho v^2$$

kde ρ je hustota vzduchu, S je priečny prierez telesa a C je koeficient závislý najmä od tvaru telesa (ale napríklad aj od rýchlosti). Predpokladajme, že skokan v skafandri s celkovou hmotnosťou 80 kg má

približne tvar valca s priemerom 0,8 m, teda s prierezom 0,5 m^2. Koeficient C pre takýto valec má hodnotu približne C = 0,8 – 1,0. Vezmeme vyššiu hodnotu C = 1, keďže skokan stabilizoval svoju polohu trením o vzduch upažením rúk (väčšie trenie).

Hustota vzduchu závisí od nadmorskej výšky. Pre nadmorské výšky do 80 km možno závislosť tlaku od výšky aproximovať vzťahom

$$p(h) = p(0)e^{-\frac{h}{H}}$$

kde $p(0)$ = 100 000 Pa je atmosférický tlak pri povrchu Zeme a koeficient H = 7400 m vyjadruje rýchlosť poklesu tlaku s výškou. Teplota vzduchu sa s výškou síce mení (vo výške asi 15 – 20 km je najnižšia a dosahuje až –70 °C), z pohľadu absolútnej teploty sú však tie zmeny menej dramatické (290 K pri povrchu Zeme, najnižšia teplota asi 220 K). Preto aj hustotu vzduchu budeme aproximovať vzťahom

$$\rho(h) = \rho(0)e^{-\frac{h}{H}}$$

kde $\rho(0)$ = 1,22 kg/m^3 je hustota vzduchu pri povrchu Zeme.

Pohybová rovnica opisujúca pád skokana má tvar:

$$m\frac{dv}{dt} = mg - F_t = mg - \frac{1}{2}CS\rho(h)v^2 = mg - \frac{1}{2}CS\rho(0)e^{-\frac{h}{H}}v^2$$
$$\frac{dv}{dt} = g - \frac{CS\rho(0)}{2m}e^{-\frac{h}{H}}v^2$$

Po dosadení konštánt dostávame:

$$\frac{dv}{dt} = 9{,}81 - 0{,}00381e^{-\frac{h}{7400}}v^2$$

Tu si treba uvedomiť, že s dobou pádu sa zmenšuje aj nadmorská výška, v ktorej sa skokan nachádza, podľa vzťahu:

$$\frac{dh}{dt} = -v$$

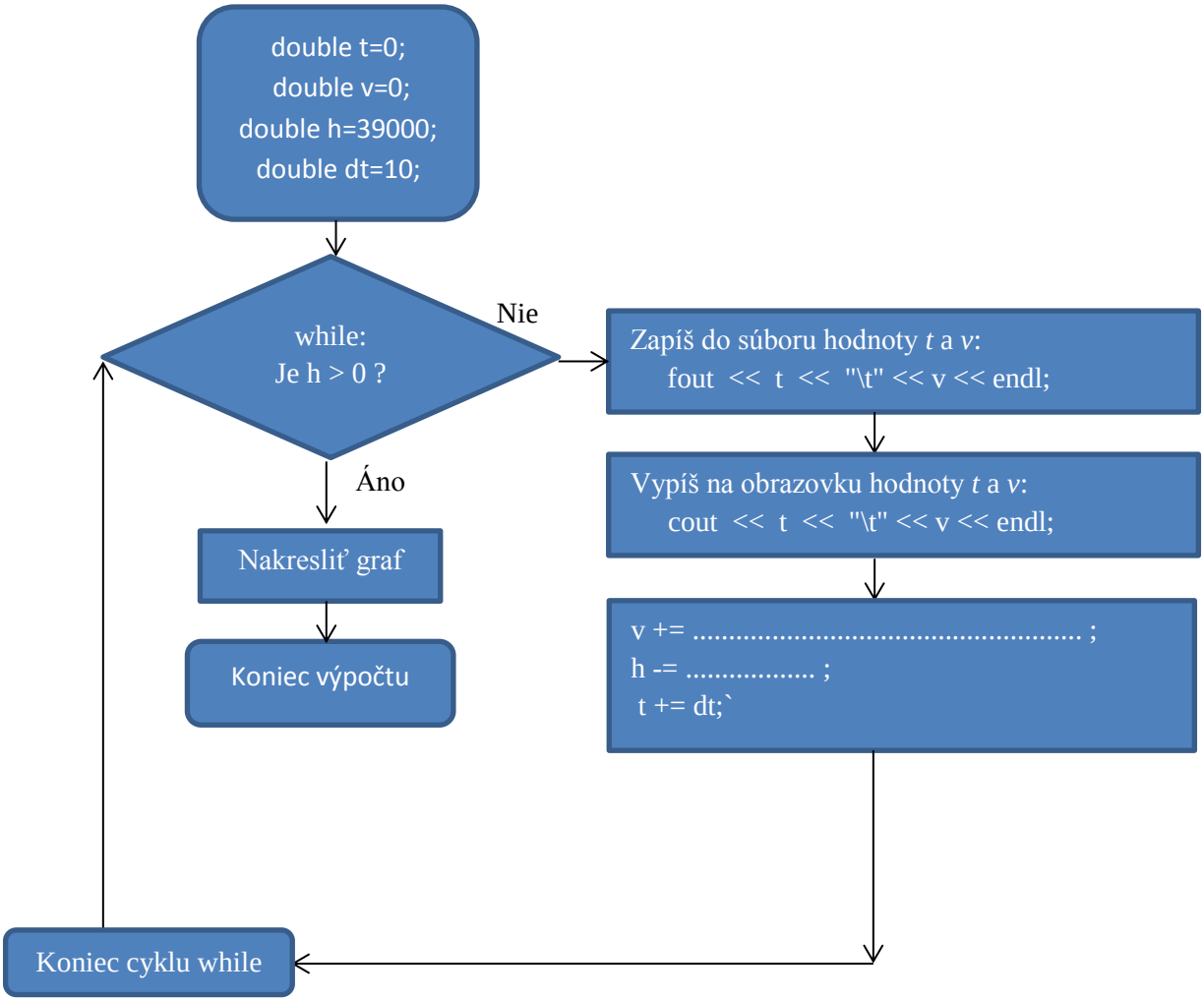
Riešenie tejto sústavy rovníc Eulerovou metódou je jednoduché. Zvolíme si pevný (malý) časový krok Δt, zvolíme počiatočné podmienky na začiatku pohybu (h = 39000 m, v = 0 m/s) a rýchlosť a výšku v nasledujúcom časovom kroku vypočítame vzťahmi:

$$v_i = v_{i-1} + (9{,}81 - 0{,}00381e^{-\frac{(h_{i-1})}{7400}}v_{i-1}^2)dt$$
$$h_i = h_{i-1} - v_{i-1}dt$$

Výpočet zastavíme po dosiahnutí nadmorskej výšky h = 0 m.

Úloha: Upravte program z cvičení tak, aby vypočítal časový vývoj rýchlosti a nadmorskej výšky skokana. Nájdite vhodnú dĺžku časového kroku. Akú najvyššiu rýchlosť skokan dosiahol a v akej to bolo výške? Akou rýchlosťou padal skokan ku koncu pádu? Koľko trval celý pád? Porovnajte to s údajmi z reálneho skoku.

Návod: Pre výpočet rýchlosti použite nasledujúci vývojový diagram. Na zodpovedanie, v akej výške dosiahol skokan maximálnu rýchlosť, upravte program tak, že do súboru zapíšete namiesto dvojíc *t*, *v* dvojice *h*, *v*. Tým získate graf *v*(*h*).



8. Šírenie chýb merania, metóda Monte Carlo, trieda `vector`

8.1. Prehľad vedomostí potrebných pre absolvovanie cvičenia

- Na prácu s veľkým množstvom čísel (zoznamom) slúži trieda `vector`, ktorá sa nachádza v knižnici `<vector>`. Jej hlavnou výhodou je, že počet prvkov tejto triedy netreba zadať v procese deklarácie, ale automaticky sa zväčšuje podľa potreby v čase behu programu. Tým sa eliminuje zbytočne prehnané alokovanie pamäte pre dáta pri kompilácii, keď ešte nevieme, koľko dát budeme reálne spracovávať.

- Trieda `vector` umožňuje automaticky kontrolovať index, takže nedovolí "použitie" neexistujúceho prvku.

- Prvkami triedy `vector` môžu byť ľubovoľné dátové typy. Typ sa určuje pri deklarácii:
```
#include <vector>
vector <double> x; //Vektor prvkov typu double
```

- K prvkom triedy `vector` sa pristupuje rovnako, ako k prvkom jednorozmerného poľa:
```
x[2]=5;
x[3]=x[2];
```

- V prípade potreby môžeme určiť počiatočnú veľkosť zoznamu a hodnoty naplniť číslom:
```
vector <float> v(100,2.1);  //prvych 100 prvkov bude mat
                               hodnotu 2,1
```
Bez uvedenia inicializačnej hodnoty obsahujú prvky zoznamu nulu.

- Doinicializovanie zoznamu:
```
v.resize(n,2.1);  //doplnenie zoznamu na rozmer n,
                    nove prvky budu 2.1
```
Funkcia sa dá použiť aj na skrátenie vektora.

- Vytvorenie kópie zoznamu:
```
vector <float> v2(v); //v2 je kopiou zoznamu v
```

- Pridanie nového prvku na koniec:
```
v.push_back(1.8);
```

- Zistenie aktuálneho počtu prvkov:
```
v.size();
```

- Počet prvkov objektu triedy `vector` môže byť taký veľký, koľko len umožňuje hardvér počítača. To môže byť viac, než dokážu pojať čísla `int` alebo `long`. Preto sa na indexovanie používa dátový typ `size_t`, ktorý dokáže pojať aj najväčšie číslo umožnené architektúrou počítača:
```
size_t  i =  1000;
```

```
v[i] = 5;
```

- Prístup k prvkom zoznamu s kontrolou hraníc:
  `v.at(279);`

- Prístup k prvkom vo vnútri zoznamu (okrem prístupu cez index) sa robí pomocou adresy prvku, t.j. cez tzv. *iterátor* (technický termín pre prácu so zoznamami). Je to v podstate adresa v pamäti, kde sa nachádza daný prvok, ale pri zväčšení iterátora o 1sa iterátor v skutočnosti zväčší o toľko bajtov, koľko zaberá jeden prvok zoznamu. Deklarácia iterátora:
  ```
  vector<float> v;
  vector<float>::iterator it=v.begin(); //ukazuje na začiatok
  ```

- Vloženie prvku dovnútra zoznamu:
  `v.insert(iterator kam, hodnota);`
  Príklad: vloženie čísla 2.1 medzi 4. a 5.pozíciu:
  `v.insert(v.begin()+4, 2.1);`

- Vymazanie prvých 5 prvkov:
  `v.erase(v.begin(), v.begin()+4);`
  Vymazanie celého zoznamu (nebude obsahovať ani jeden prvok):
  `v.clear();`

- Rýchle (metódou quick sort) utriedenie zoznamu (alebo jeho časti určenej iterátormi) sa robí funkciou `sort()`, ktorá sa nachádza v knižnici `<algorithm>`:
  ```
  ...
  #include <algorithm>
  sort(v.begin(),v.end()); // utriedi sa celý zoznam
  ```

- Otočenie poradia prvkov:
  `reverse(v.begin(),v.end());`

- Maximálna a minimálna hodnota:
  `min_element(v.begin(),v.end());` – vráti iterátor na najmenší prvok zo zoznamu
  `max_element(v.begin(),v.end());` – vráti iterátor na najväčší prvok zo zoznamu

- Do volaných funkcií nie je vhodné posielať celý zoznam (pomalé a náročné na pamäť), používajú sa referencie na zoznam (rovnako ako u jednorozmerného poľa):
  `vector <float> &v;`

- Na vygenerovanie náhodného čísla slúži funkcia `rand()` z knižnice `<cstdlib>`. Výsledkom je náhodné celé číslo z intervalu 0 až `RAND_MAX`. Hodnota `RAND_MAX` závisí od hardvéru a softvéru počítača:
  ```
  #include <cstdlib>
  ...
  int n;
  n=rand();
  ```

- Pri každom spustení počítača sa generujú postupne tie isté náhodné čísla. Inicializáciu generátora náhodných čísel umožňuje funkcia `srand(číslo)`. Ak argument funkcie `srand()` odvodíme od aktuálneho času (funkcia `time` v knižnici `<ctime>`) , pri každom spustení programu sa budú generovať iné náhodné čísla:

```
#include <cstdlib>
#include <ctime>
...
srand(time(NULL));
x = rand();
```

8.2. Neistoty merania a ich šírenie

Na praktikách potrebujete pravidelne vyhodnocovať chyby merania. Na tomto cvičení sa zameriame na počítačové spracovanie tejto úlohy.

Namerané hodnoty spravidla nie sú presné, ale náhodne odchýlené od správnej hodnoty. Najčastejšie sa stretávame s dvoma prípadmi:

1. Opakovaním merania získavame rôzne hodnoty náhodne rozmiestnené okolo správnej (strednej) hodnoty. Obvykle namerané hodnoty pochádzajú z Gaussovho rozdelenia. V takom prípade správnu hodnotu odhadujeme z n takýchto nameraných hodnôt x_i pomocou aritmetického priemeru:

   $$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

   a chybu merania (rozptyl nameraných bodov) odhadujeme štandardnou odchýlkou

   $$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

   Pozrite si tiež cvičenie 4, kde sme sa zaoberali charakteristikami štatistických súborov.

2. Opakovaním merania získavame stále tú istú hodnotu, tá je však odchýlená od správnej hodnoty kvôli presnosti meracieho prístroja. Od výrobcu máme iba garantovanú presnosť prístroja (maximálnu odchýlku údaju od skutočnej hodnoty), ktorú označíme písmenom a. Keďže skutočná hodnota môže byť kdekoľvek okolo nameranej hodnoty (najviac ale vzdialená o a), situáciu modelujeme rovnomerným rozdelením. Štandardnou odchýlkou rovnomerného rozdelenia s polšírkou a je výraz:

   $$\sigma = \frac{a}{\sqrt{3}} = \frac{a}{1{,}73}$$

V praxi sa okrem štandardnej odchýlky (*štandardnej neistoty merania*) používa aj interval, v ktorom sa skutočná hodnota nachádza s vysokou (napr. 95%) pravdepodobnosťou. Takýto interval sa označuje *rozšírená neistota merania*.

Často veličinu meriame nepriamo, t. j. najprv priamo zmeriame niekoľko rôznych veličín x_1, x_2, x_3 ... a hľadanú veličinu y vypočítame zo vzťahu

$$y = f(x_1, x_2, x_3, ..)$$

Tu vyvstáva kľúčová otázka: **Ak poznáme hodnoty a štandardné odchýlky dielčích veličín, aká je stredná hodnota a štandardná odchýlka veličiny *y*?**

Úloha sa rieši rôznymi postupmi, často je postup založený napríklad linearizácii funkcie. Tá je opodstatnená, ak chyby merania sú natoľko malé, aby sa v ich rozsahu dala funkcia aproximovať Taylorovým rozvojom do 1. rádu. Výhodou linearizácie je, že hľadané vzťahy sa dajú vyjadriť analyticky.

Alternatívnou metódou je nič neaproximovať a situáciu analyzovať generovaním veľkého množstva sérií náhodných čísel (x_1, x_2, x_3, ...) z príslušných rozdelení (Gaussovho alebo rovnomerného). Tak získame obrovskú sériu čísel *y*. Z nich ľahko vypočítame aritmetický priemer (odhad skutočnej hodnoty), štandardnú odchýlku (štandardnú neistotu odhadu) alebo po utriedení dát podľa veľkosti aj interval, v ktorom sa nachádza napríklad 95% hodnôt *y*, teda rozšírenú neistotu merania. Keďže táto metóda je založená na generovaní náhodných čísel, patrí do skupiny metód označovaných ako Monte Carlo.

8.3. Generovanie náhodných čísel, štandardná odchýlka, interval 95%

Na generovanie náhodných celých čísel slúžia funkciae `rand()` a `srand()`, ktoré sa nachádzajú v knižnici `<cstdlib>`. Z celých čísel generovaných funkciou `rand()` môžeme jednoduchou matematickou úpravou získať náhodné desatinné čísla s rovnomerným rozdelením. Pre potreby spracovania experimentálnych dát však budeme potrebovať náhodné čísla pochádzajúce aj z Gaussovho rozdelenia. Generovanie čísel z iných rozdelení pomocou rovnomerného rozdelenia je založené na znalosti inverznej funkcii k rozdeľovacej funkcii. V prípade Gaussovho rozdelenia je možné použiť približný tvar tejto inverznej funkcie, aby sa dal vyjadriť analyticky.

Nasledujúca funkcia vráti náhodné číslo s rozdelenia určeného strednou hodnotou (prvý parameter funkcie), štandardnou odchýlkou (druhý parameter funkcie) a typom rozdelenia (tretí parameter: 0 – rovnomerné, 1 – Gaussovo):

```
double Random(double mean, double sigma, int typroz)
{
double ran,xran;
double u,v,x;
switch(typroz)
    {
     case 0: //vyuzijeme standardny generator
             ran=((float)rand()+0.5)/(float)(RAND_MAX+1.0);
             xran=2.0*(ran-0.5);
              return mean + xran*sigma*1.73;
     case 1: //vyuzijeme Box-Mullerovu metodu
             u = ((float)rand()+0.5)/(float)(RAND_MAX+0.5);
             v = ((float)rand()+0.5)/(float)(RAND_MAX+0.5);
             x = sqrt(-2*log(u))*sin(2*M_PI*v);
             return mean+x*sigma;
      default: return 0;
   }
}
```

Úloha: Napíšte program, v ktorom vygenerujete veľké množstvo čísel typu `double` z Gaussovho rozdelenia so strednou hodnotou 1 a štandardnou odchýlkou 2. Na ukladanie čísel použite objekt `vector`. Potom z čísel vypočítajte aritmetický priemer

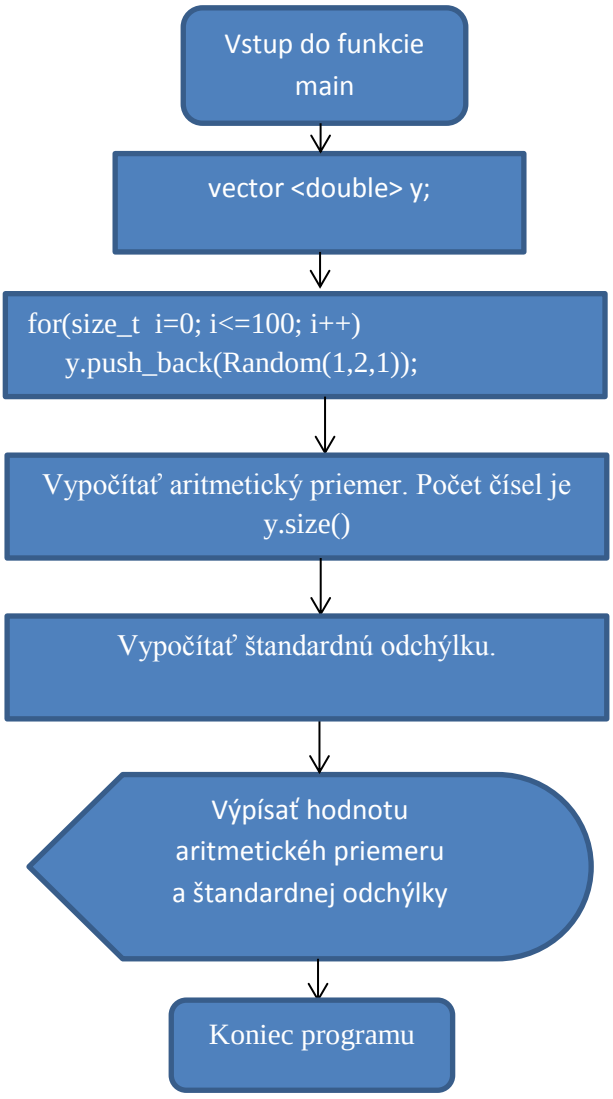
$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

a štandardnú odchýlku

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

a porovnajte ich s presnými hodnotami 1 a 2. Aké množstvo čísel treba, aby sa od presných hodnôt nelíšili viac než o 0,05?

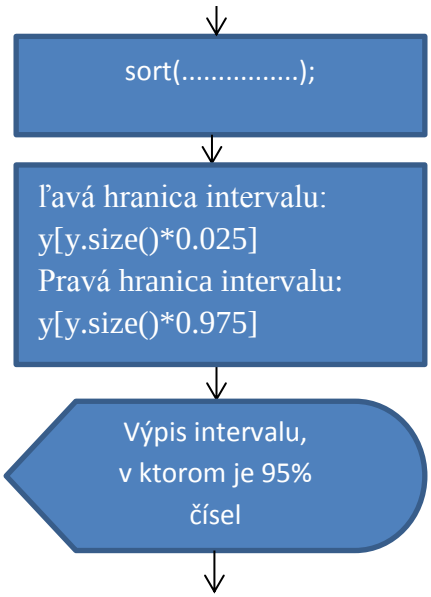
Návod: Použite rámcovú blokovú schému z nasledujúceho obrázku.



Na určenie intervalu, v ktorom leží 95% čísel, musíme čísla najprv utriediť. Použijeme na to funkciu `sort()` z knižnice `<algorithm>`.

Úloha: Doplňte program o nájdenie intervalu, v ktorom leží 95% čísel. Koľko čísel treba vygenerovať, aby sa hranice intervalu nelíšili od teoretickej hodnoty <-2,92; 4.92> o viac než 0,1?

Návod: Použite nasledujúci blokový diagram.



8.4. Neistota nepriamo meranej veličiny $y = e^x$

Pri výpočte neistoty nepriamo meranej veličiny budeme vychádzať z programu z časti 2. Jediným rozdielom bude, že si nebudeme odkladať náhodné čísla x, ale rovno funkčné hodnoty $f(x)$.

Úloha. Upravte program tak, že vyriešite nasledujúci problém. Hodnotu x sme zmerali niekoľkokrát a z nameraných hodnôt sme aritmetickým priemerom vypočítali strednú hodnotu (1,21) a štandardnú odchýlku aritmetického priemeru (0,18). Hľadáme nepriamo meranú veličinu y, pre ktorú platí $y = e^x$. Predpokladáme, že aritmetický priemer pochádza z Gaussovho rozdelenia s uvedenou strednou hodnotou a štandardnou odchýlkou. Nájdite strednú hodnotu veličiny y, jej štandardnú a rozšírenú (95%) neistotu. Porovnajte výsledky s hodnotami získanými linearizáciou: 3,353 ; 0,604 ; < 2,17 ; 4,54 >.

Úloha. Zmeňte situáciu tak, že sme merali 10x presnejšie, takže štandardná odchýlka bude iba 0,018. Porovnajte získané výsledky s hodnotami získanými linearizáciou: 3,353 ; 0,0604 ; < 3,23 ; 3,45 >. Vysvetlite výsledky získané v oboch úlohách.

8.5. Neistota nepriamo meranej veličiny $y = a.b^2/c^3$

Pri výpočte neistoty nepriamo meranej veličiny budeme vychádzať z programu z časti 3. Jediným rozdielom bude, že nebudeme generovať jedno náhodné číslo *x*, ale tri: *a*, *b*, *c*. Každé číslo pritom vygenerujeme z rozdelenia podľa nameranej hodnoty a chyby merania.

Úloha. Upravte program tak, že vyriešite nasledujúci problém. Hodnotu *a* sme zmerali niekoľkokrát a z nameraných hodnôt sme aritmetickým priemerom vypočítali strednú hodnotu (2,31) a štandardnú odchýlku aritmetického priemeru (0,072). Hodnotu *b* sme takisto zmerali niekoľkokrát a z nameraných hodnôt sme aritmetickým priemerom vypočítali strednú hodnotu (1,62) a štandardnú odchýlku aritmetického priemeru (0,044). Predpokladáme, že oba aritmetické priemery pochádzajú z Gaussovho rozdelenia s uvedenými strednými hodnotami a štandardnými odchýlkami. Veličinu *c* sme zmerali jedenkrát (3,52) s maximálnou chybou 0,035. Preto predpokladáme, že nameraná hodnota pochádza z rovnomerného rozdelenia so štandardnou odchýlkou 0,035/1,73 = 0,020. Nájdite strednú hodnotu veličiny *y*, jej štandardnú a rozšírenú (95%) neistotu.

Kontrola: 0,139 ; 0,009 ; < 0,123 ; 0,158 >.

8.6. Domáca úloha – spracovanie merania viskozity oleja z pohybu guličky

Viskozitu kvapaliny možno zmerať pozorovaním pádu kovovej guľôčky v nej. Vplyvom trenia sa rýchlosť pádu guľôčky ustáli na konštantnej hodnote. Jej zmeraním možno určiť viskozitu kvapaliny.
Pre ustálenú rýchlosť guľôčky platí vzťah

$$6K\pi\,\eta rv = \frac{4}{3}\pi r^3(\rho_z - \rho_k)g$$

kde r je polomer guľôčky, η je koeficient viskozity, g je gravitačné zrýchlenie, ρ_z je hustota materiálu guľôčky (v našom prípade železa), ρ_k je hustota kvapaliny a K empirická konštanta, ktorá sa rovná

$$K = 1 + \left(\frac{9r}{2D}\right) + \left(\frac{9r}{2D}\right)^2$$

kde D priemer valca s kvapalinou, v ktorom robíme pokus.

Rýchlosť guľôčky môžeme určiť z doby pádu guľôčky t medzi dvoma značkami na valci, pričom vzdialenosť značiek je l = 0,5 m. Vzdialenosť značiek na valci sme určili pravítkom s maximálnou neistotou 0,5 mm. Pokus sme zopakovali viackrát, pre meranie času sme použili mechanické stopky, získali sme priemernú hodnotu t = 4,0 s so štandardnou neistotou 0,1 s.

Hustotu kvapaliny (oleja) sme získali z tabuliek ρ_k = 800 kg.m^{-3} pričom tabuľky uvádzajú maximálnu neistotu tohto údaju 20 kg.m^{-3}. Hustotu guľôčky sme určili meraním jej polomeru mikrometrom (objem) a jej hmotnosti analytickými váhami. Polomer sme určili ako r = 1,95 mm s maximálnou neistotou danou presnosťou mikrometra 0,02 mm. Hmotnosť guľôčky je m = 0,242 3 g s maximálnou neistotou danou presnosťou váh 0,000 1 g. Priemer valca sme určili meraním pravítkom D = 5,0 cm s maximálnou neistotou 0,5 mm. Gravitačné zrýchlenie v mieste merania je g = 9,808 88 m.s^{-2} a je známe s presnosťou vysoko prevyšujúcou ostatné merané veličiny.

Dosadením priamo meraných veličín do uvedeného vzťahu dostávame

$$\eta = \frac{\left(m - \frac{4}{3}\pi r^3 \rho_k\right).g.t}{6\pi r l\left[1 + \left(\frac{9r}{2D}\right) + \left(\frac{9r}{2D}\right)^2\right]}$$

a) Určte hodnotu koeficientu viskozity kvapaliny.

b) Nájdite štandardnú neistotu v určení hodnoty koeficientu viskozity.

d) Určte rozšírenú (95%) neistotu v určení koeficientu viskozity.

9. Fourierove rady, kreslenie priebehu funkcií

9.1. Prehľad vedomostí potrebných na absolvovanie cvičenia

- Hodnota Ludolphovho čísla je definovaná v knižnici `<cmath>` ako konštanta `M_PI`. Príklad:

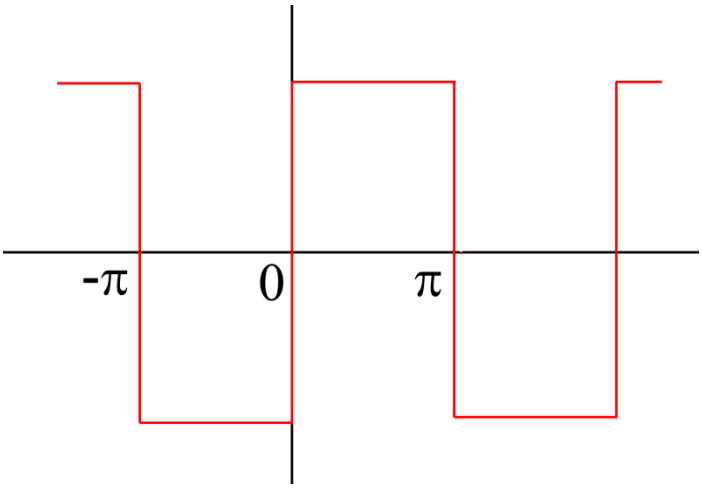
```
#include <cmath>
...
x = 2*M_PI;
```

9.2. Fourierove rady - úvod

Periodické funkcie s frekvenciou *f* možno vyjadriť v tvare súčtu sínusoviek a kosínusoviek s rôznymi kmitočtami, ktoré sú celočíselným násobkom kmitočtu *f*.

Napríklad obdĺžnikový priebeh (obr.1) možno vyjadriť v tvare súčtu sínusoviek

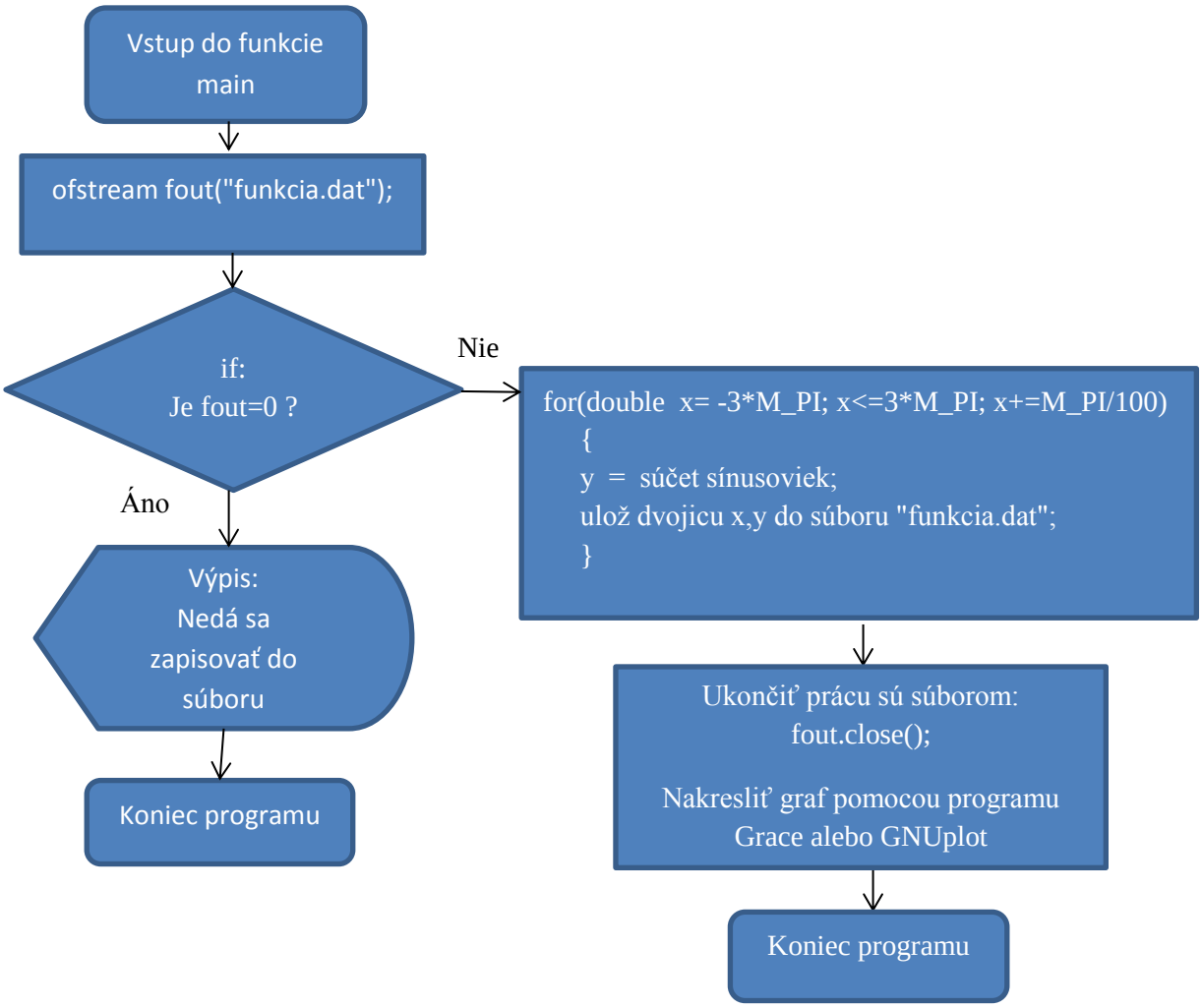
$$\sin(x)+\frac{1}{3}\sin(3x)+\frac{1}{5}\sin(5x)+\frac{1}{7}\sin(7x)+\cdots$$



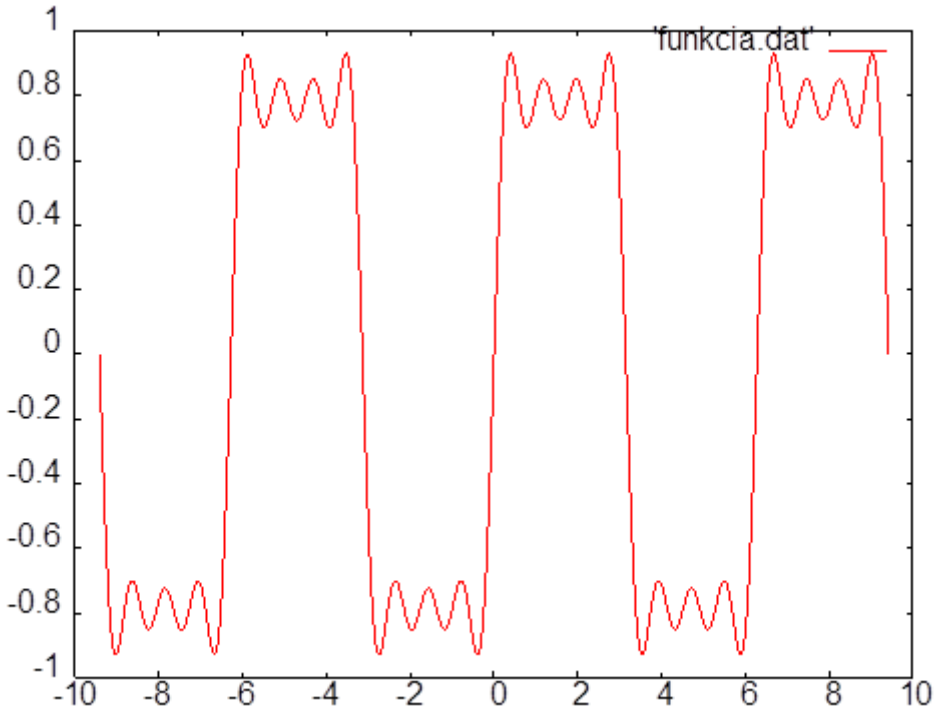
Obr.1. Periodický "obdĺžnikový" priebeh

Úloha: Napíšte program, v ktorom vypočítate výraz (súčet) uvedený vyššie po člen sin(7*x*) vrátane. Hodnotu *x* meňte v rozsahu <-3π, 3π> s jemným krokom π/100 . Dvojice *x*, *y* zapíšte do súboru "funkcia.dat" tak, že čísla *x* a *y* budú oddelené tabulátorom a každá dvojica bude na samostatnom riadku. Nakoniec z uložených dvojíc nakreslite graf.

Návod. Využite svoje skúsenosti s podobnými úlohami na predchádzajúcich cvičeniach a blokový vývojový diagram z nasledujúceho obrázku. Aby ste nemuseli do programu vpisovať hodnotu Ludolfovho čísla, použite hodnotu `M_PI`, ktorá je definovaná v knižnici `<cmath>`.



Kontrola. Mali by ste dostať takýto obrázok:



9.3. Fourierove rady – všeobecný tvar. Príklady funkcií.

Všeobecný tvar radu pre výpočet hodnoty periodickej funkcie s periódou 2π je:

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos(nx) + b_n \sin(nx) \right)$$

kde a_0 je priemerná hodnota funkcie, a_n a b_n sú tzv. Fourierove koeficienty. V prípade, že funkcia je nepárna, je súčtom iba sínusov, keďže kosínus je párna funkcia. Naopak, ak funkcia je párna, v súčte sa nevyskytujú sínusy, iba kosínusy. Obdĺžniková funkcia na obrázku 1 je nepárna, preto je súčtom sínusov.

Úloha: Upravte program tak, že v cykle od $n = 1$ až po 10 (alebo po iné číslo) sčítate rad zodpovedajúci obdĺžnikovému priebehu

$$f(x) = \sum_{n=1}^{10} \frac{1}{2n-1} \sin((2n-1)x)$$

Zistite, koľko členov musíme sčítať, aby sa súčet radu prakticky nelíšil od skutočného obdĺžnikového priebehu.

Návod. Do už hotového cyklu `for(double x=-3*M_PI;x<=3*M_PI;x+=M_PI/100)` vložte vnútorný cyklus `for(int n=1;n<=10;n++)`, v ktorom do výsledku (súčtu radu) budete pripočítavať jednotlivé členy `sin((2*n-1)*x)/(2*n-1)`. Nezabudnite si premennú na odkladanie súčtu pred vstupom do cyklu vynulovať!

Teraz si už môžeme skúsiť nakresliť priebehy rôznych funkcií daných ich Fourierovým radom. V programe nám však ešte chýba pripočítať člen a_0 (neobsahuje sínus ani kosínus).

Úloha: Upravte program tak, že k súčtu radu pripočítate ešte konštantný člen a_0. Potom nakreslite priebehy periodických funkcií daných ich Fourierovými radmi uvedenými nižšie. Pozrite si, ako vyzerá výsledok, keď sčítate menej alebo viac členov.

$$f_1(x) = \frac{4}{\pi} \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{(2n-1)^2} \sin((2n-1)x)$$

$$f_2(x) = \sum_{n=1}^{\infty} \cos((2n-1)x)$$

$$f_3(x) = \frac{8}{\pi} \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{(2n-1)^3} \sin((2n-1)x)$$

$$f_4(x) = \frac{2}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} \sin(nx)$$

$$f_5(x) = \frac{\pi}{8} + \frac{2}{\pi}\sum_{n=1}^{\infty} \frac{1-\cos\left(\frac{n\pi}{2}\right)}{n^2}\cos(nx)$$

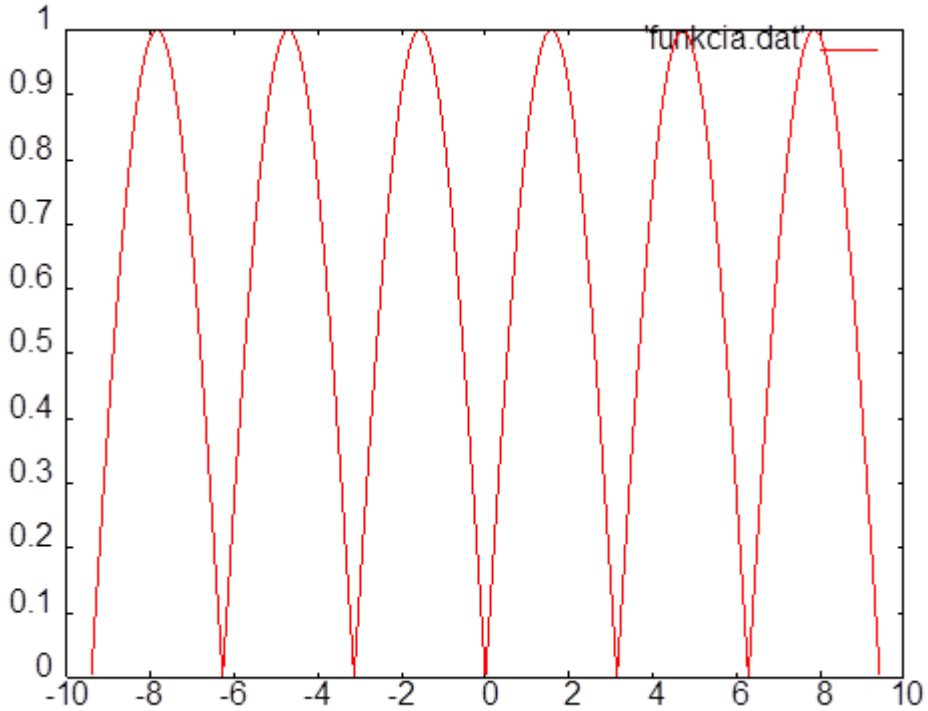
9.4. Domáca úloha – všeobecné vlastnosti Fouriérových radov

V nasledujúcich dvoch Fourierovych radoch je chyba vo vzorcoch (v každom rade je zámerne urobená iba jedna chyba). Naprogramujte podľa vzorca pre Fourierov rad vykreslenie grafu funkcie a pokúste sa zistiť, v ktorej časti vzorca by mohla byť chyba, prípadne sa ju pokúste aj odstrániť.

Prvý rad má predstavovať periodicky sa opakujúci úsek funkcie $\sin(x)$, presnejšie $f_6(x) = |\sin(x)|$:

$$f_6(x) = \frac{1}{\pi} - \frac{4}{\pi}\sum_{n=1}^{\infty} \frac{1}{4n^2 - 1}\cos(2nx)$$

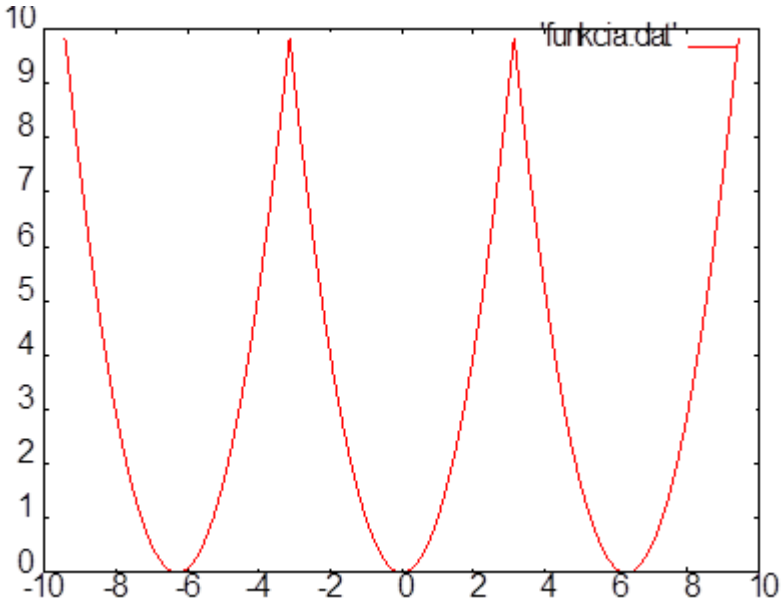
Správny graf tejto funkcie vyzerá takto:



Správny graf funkcie $f_6(x)$

Druhý rad má predstavovať periodicky sa opakujúci úsek funkcie x^2 z intervalu <-π, π>:

$$f_7(x) = \frac{\pi^2}{3} + 4\sum_{n=1}^{\infty} \frac{(-1)^n}{n^2} \sin(nx)$$



Správny graf funkcie $f_7(x)$

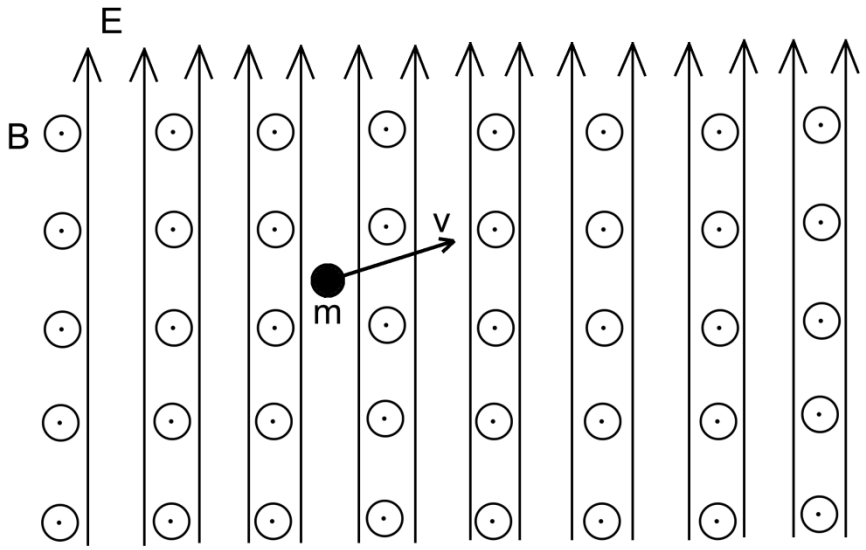
10. Pohyb elektrónu v skríženom elektrickom a magnetickom poli, balistická krivka.

10.1. Vedomosti potrebné na absolvovanie cvičenia

Na tomto cvičení vystačíte s vedomosťami získanými na predchádzajúcich cvičeniach.

10.2. Úvod

Na tomto cvičení budeme riešiť pohyb nabitej častice v skríženom elektrickom a magnetickom poli. Homogénne elektrické pole má smer osi *y* (obr.1), homogénne magnetické pole je na neho kolmé a na obr. 1 vystupuje z nákresne smerom k pozorovateľovi. Častica má hmotnosť *m* a elektrický náboj *q*.



Obr. 1. Častica sa pohybuje v elektrickom a magnetickom poli

10.3. Pohybové rovnice

Pohyb častice rozložíme na vodorovnú zložku (smer osi *x*) a vertikálnu zložku (smer osi *y*)

Elektrické pole pôsobí iba v smere osi *y*:

$$m\frac{dv_y}{dt} = qE$$

Smer magnetickej sily je daný vektorovým súčinom:

$$\mathbf{F}_m = q\mathbf{v}\times\mathbf{B}$$

odkiaľ dostávame

$$m\frac{dv_x}{dt} = qv_y B$$
$$m\frac{dv_y}{dt} = -qv_x B$$

Spojením uvedených vzťahov dostaneme

$$\frac{dv_x}{dt} = \frac{q}{m}v_y B$$
$$\frac{dv_y}{dt} = \frac{q}{m}(E - v_x B)$$

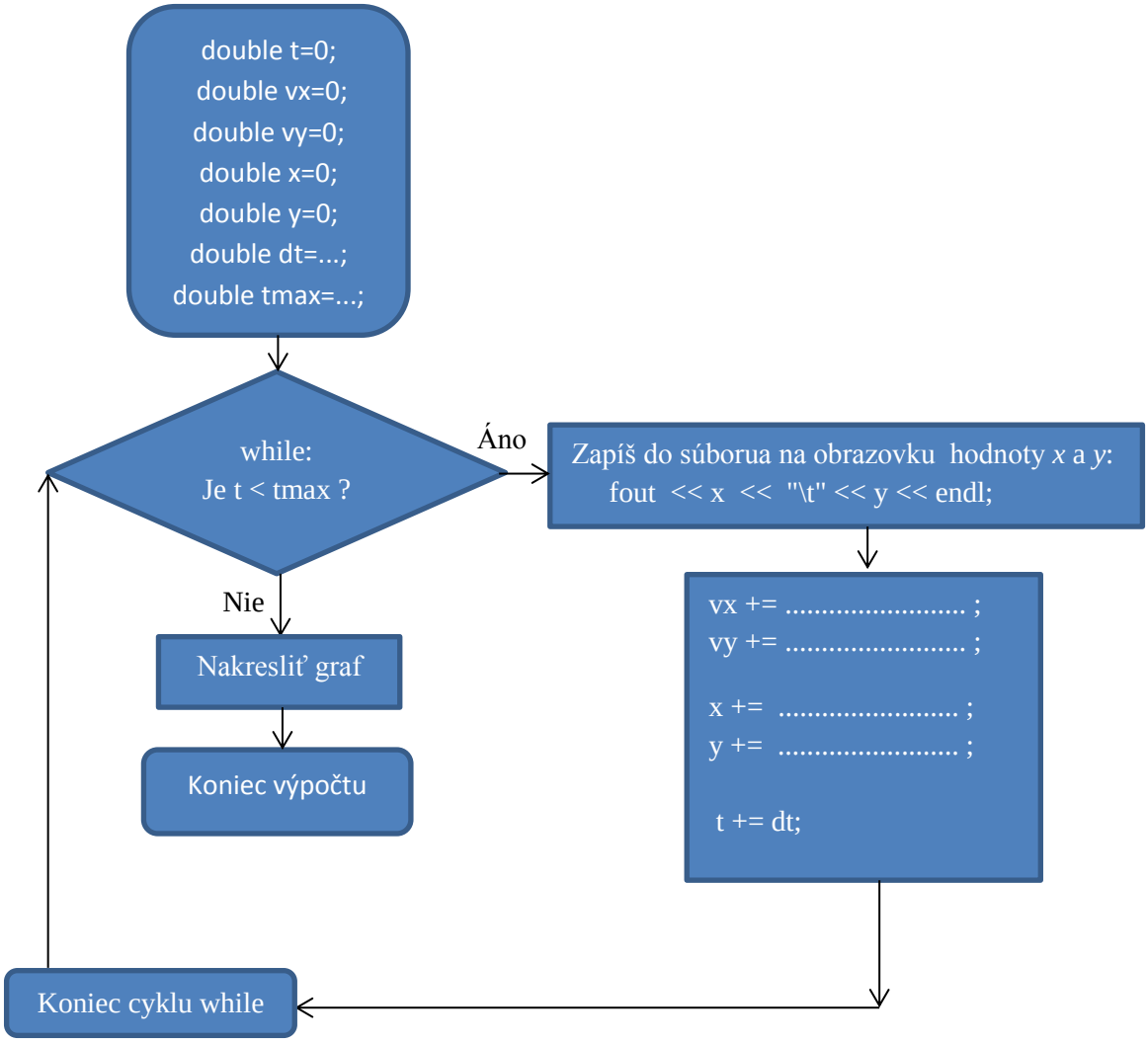
Túto sústavu diferenciálnych rovníc budeme riešiť Eulerovou metódou, s ktorou sme sa oboznámili v cvičení 7:

$$v_x(t_i) = v_x(t_{i-1}) + \frac{q}{m}v_y(t_{i-1})B\Delta t$$
$$v_y(t_i) = v_y(t_{i-1}) + \frac{q}{m}(E - v_x(t_{i-1})B)\Delta t$$
$$x(t_i) = x(t_{i-1}) + v_x(t_{i-1})\Delta t$$
$$y(t_i) = y(t_{i-1}) + v_y(t_{i-1})\Delta t$$

V tomto cvičení budeme počítať pohyb protónu (m = 1,67 . 10^{-27} kg, q = 1,602 . 10^{-19} C, q/m = 9,59 . 10^7 C/kg) v elektrickom poli E = 100 V/m a magnetickom poli B = 0,1 T.

Úloha: Napíšte program, ktorý Eulerovou metódou vypočíta pohyb protónu v skríženom elektrickom a magnetickom poli podľa zadania. Na začiatku pohybu predpokladajte rýchlosť protónu v_x = 0 m/s, v_y = 0 m/s. Do súboru "poloha.dat" ukladajte dvojice vypočítaných hodnôt x a y oddelené tabulátorom, každá dvojica na novom riadku. Nájdite vhodný časový krok Δt a celkový čas výpočtu t_{max} tak, aby ste dobre videli tvar dráhy protónu. Po skončení výpočtu nakreslite dráhu programom Grace alebo GNUplot.

Návod. Vývojový diagram výpočtového jadra programu je na nasledujúcom obrázku.



10.4. Pohyb protónu v elektrickom poli

Úloha: Dajte protónu počiatočnú rýchlosť v_x = 1000 m/s a nastavte *B* = 0 T (*E* ponechajte 100 V/m). Pozrite si dráhu protónu a porovnajte ju s tým, čo ste očakávali.

10.5. Pohyb protónu v magnetickom poli

Úloha: Dajte protónu počiatočnú rýchlosť v_x = 1000 m/s a nastavte *E* = 0 V/m (*B* ponechajte 0,1 T). Pozrite si dráhu protónu a porovnajte ju s tým, čo ste očakávali.

10.6. Pohyb protónu rýchlosťou $v_x = E/B$

Úloha. Z rovníc vidíme, že ak sa protón pohybuje rýchlosťou $v_x = E/B = 1000$ m/s, elektrická a magnetická sila sa úplne vykompenzujú. Overte to vaším programom. Pozorujte dráhu protónu, aj ak je rýchlosť o trochu nižšia alebo vyššia.

10.7. Pohyb protónu rýchlosťou $v_x = 10\,000$ m/s

Úloha. Pozrite si dráhu protónu, ak jeho počiatočná rýchlosť bola 10 000 m/s. Aký tvar dráhy očakávate, ak bude počiatočná rýchlosť protónu ešte vyššia?

10.8. Domáca úloha – balistická krivka

Upravte váš program tak, aby počítal dráhu projektilu ovplyvnenú odporom vzduchu. Predpokladajte, že projektilom je oceľová delová guľa (hustota 7800 kg/m^3) s polomerom 5 cm, ktorá opustí hlaveň dela rýchlosťou $v_0 = 300$ m/s.

1. Namiesto elektrickej sily pôsobí na projektil gravitačná sila, ale v zápornom smere osi y:

   $$F_g = -mg$$

2. Na guľu pôsobí tiež odporová sila vzduchu

   $$F_t = \frac{1}{2} C\rho\pi r^2 v^2 = \frac{1}{2} C\rho\pi r^2 (v_x^2 + v_y^2), \quad C = 0{,}47$$

   kde r je polomer gule a $\rho = 1{,}2$ kg/m^3 je hustota vzduchu. Táto sila zmenšuje obe zložky rýchlosti gule. Preto najprv musíme vypočítať uhol letu projektilu vzhľadom k vodorovnej rovine:

   $$\tan\varphi = \frac{v_y}{v_x}, \quad \varphi = \arctan\frac{v_y}{v_x}$$

   Priemety odporovej sily do smerov x a y sú:

   $$F_{t,x} = -\frac{1}{2} C\rho\pi r^2 (v_x^2 + v_y^2)\cos\varphi$$
   $$F_{t,y} = -\frac{1}{2} C\rho\pi r^2 (v_x^2 + v_y^2)\sin\varphi$$

3. Výsledná sila pôsobiaca na projektil teda je

   $$F_x = -\frac{1}{2} C\rho\pi r^2 (v_x^2 + v_y^2)\cos\varphi$$
   $$F_y = -\frac{1}{2} C\rho\pi r^2 (v_x^2 + v_y^2)\sin\varphi - mg$$

4. Odtiaľ pre zmenu zložiek rýchlosti vplyvom trecej sily dostávame

$$\frac{dv_x}{dt} = -\frac{1}{2m} C\pi r^2 (v_x^2 + v_y^2)\cos\varphi$$
$$\frac{dv_y}{dt} = -\frac{1}{2m} C\pi r^2 (v_x^2 + v_y^2)\sin\varphi - g$$

5. Počiatočné hodnoty rýchlosti vypočítajte z rýchlosti v_0 pomocou uhla výstrelu (deklinácie) δ vzťahmi:

$$v_x(0) = v_0 \cos\delta$$
$$v_y(0) = v_0 \sin\delta$$

6. Výpočet dráhy projektilu ukončite, keď guľa padne na zem ($y = 0$).

Úloha.

Pod akým uhlom δ treba guľu vystreliť, aby doletela čo najďalej? Porovnajte tento uhol s prípadom, keď na guľu nepôsobí trecia sila (45^o).

Aký je maximálny dostrel dela?

Aký vplyv má teplota vzduchu na presnosť zásahu? Predpokladajte, že teplota sa zmenila z 20 ^{o}C (hustota vzduchu 1,2 kg/m^3) na 0 ^{o}C (hustotu vypočítajte zo stavovej rovnice). O koľko sa posunulo miesto zásahu?

11. Difrakcia svetla, Huyghensov princíp, komplexné čísla.

11.1. Prehľad vedomostí potrebných na absolvovanie cvičenia

- Rozšírenie matematických operácii na komplexné čísla poskytuje knižnica `<complex>`. Komplexné čísla môžu mať rôzny dátový typ, ktorý sa definuje pri deklarácii komplexného čísla:

```
#include <complex>
...
complex <float> x; //x je komplexné číslo typu float
```

- Komplexné číslo možno naplniť hodnotou už pri inicializácii:

```
complex <double> x(2,3); //číslo x je komplexné číslo
                            s hodnotou 2 + 3i
```

- Reálnu a imaginárnu časť sprístupňujú funkcie `real()` a `imag()`. Môžu sa použiť na pravej aj ľavej strane priradenia:

```
complex <double> c;
real(c)=2.5;
imag(c)=2*real(c);
```

- Užitočné funkcie:
  `abs(c)` – modul (amplitúda) komplexného čísla
  `norm(c)` – norma (druhá mocnina modulu)
  `arg(c)` – fáza
  `conj(c)` – komplexne združené číslo
  `sin(c), cos(c), tan(c), exp(c), ...`

11.2. Skladanie svetelných vĺn

Okamžitú hodnotu sínusovej vlny s amplitúdou A a uhlovou frekvenciou ω, šíriacej sa v smere x, môžeme opísať vzťahom

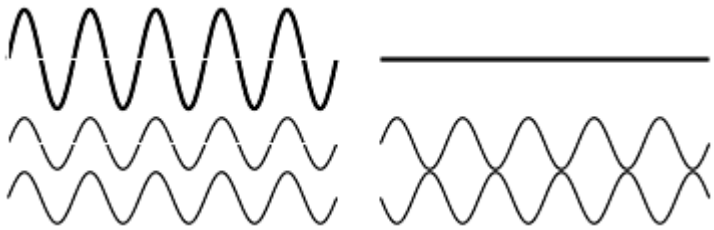
$$Ae^{i(\omega t - kx)}, \quad k = \frac{2\pi}{\lambda}$$

kde λ je vlnová dĺžka vlny. Ak do nejakého miesta x dorazia súčasne dve vlny z rôznych vzdialeností x_1 a x_2, ich amplitúdy sa sčítajú:

$$A_1 e^{i(\omega t - kx_1)} + A_2 e^{i(\omega t - kx_2)}$$

pričom sme predpokladali, že obe vlny majú rovnakú frekvenciu a vyrazili zo zdroja v rovnakej fáze (člen ωt je u oboch vĺn rovnaký).

Ak obe vlny majú rovnakú frekvenciu, potom v závislosti od toho, v akej fáze vlny dorazili, sa môžu sčítať (konštruktívna interferencia) alebo odčítať (deštruktívna interferencia):



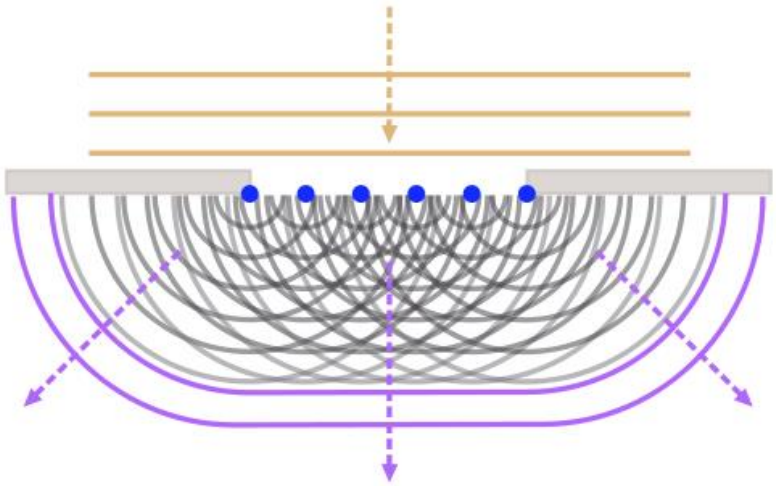
Konštruktívna (vľavo) a deštruktívna (vpravo) interferencia
Zdroj: Wikipédia

Oko a fotografický prístroj však vnímajú *časovo priemernú hodnotu energie* dopadajúceho svetla (frekvencia svetla je obrovská – v ráde 10^{14} Hz). Preto jas (intenzita) obrazca bude úmerný druhej mocnine súčtu amplitúd, teda druhej mocnine modulu:

$$I = \frac{1}{2}\left|A_1 e^{-ikx_1} + A_2 e^{-ikx_2}\right|^2$$

11.3. Huyghensov - Fresnelov princíp

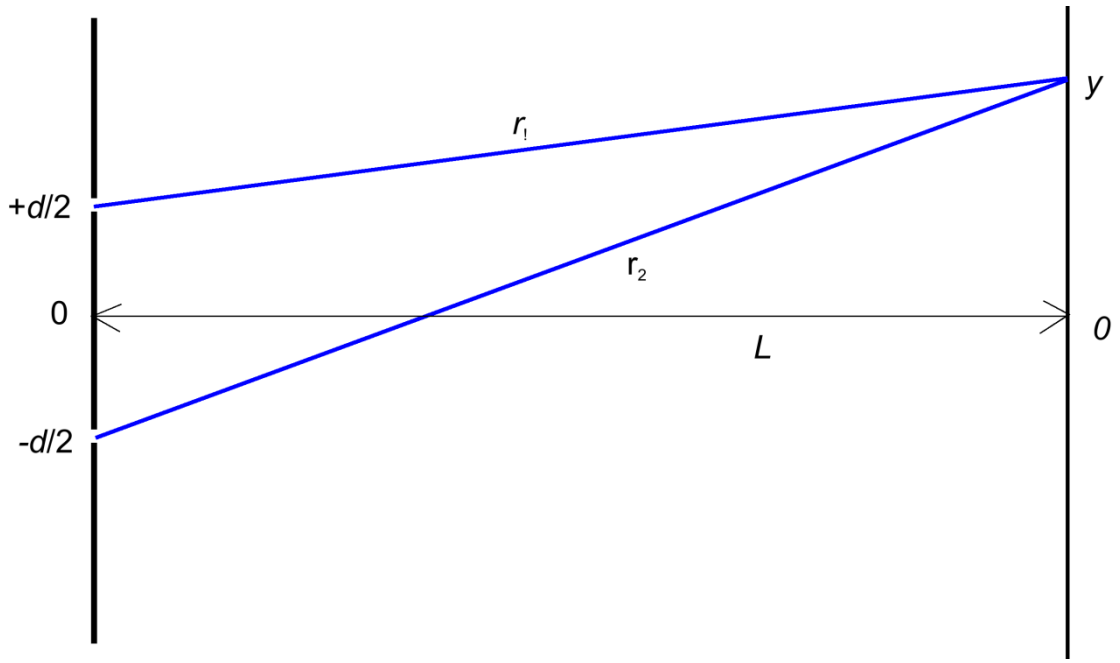
Christian Huyghens našiel jednoduchý postup, ako vypočítať šírenie svetelných vlnoplôch. Stačí si predstaviť body na vlnoploche (vlnoplocha je geometrické miesto bodov s rovnakou fázou vlny) ako nové bodové zdroje vĺn, z ktorých sa šíria nové kruhové vlnoplochy (všetkými smermi). Na nasledujúcom obrázku je ilustrovaný dopad rovinnej vlny na otvor v tienidle, kde vlnu za prekážkou získame sčítaním kruhových vlniek vychádzajúcich z jednotlivých bodov otvoru (z čela tej časti vlnoplochy, ktorá prejde otvorom):



Huyghensov – Fresnelov princíp
Zdroj: Wikipédia

11.4. Interferencia vĺn na dvojštrbine

Uvažujme dve veľmi tenké štrbiny vzdialené od seba o hodnotu *d* (napríklad 60 μm). Vo vzdialenosti *L* od štrbín (napríklad 1m) sa nachádza tienidlo, na ktorom pozorujeme interferenčný obraz:

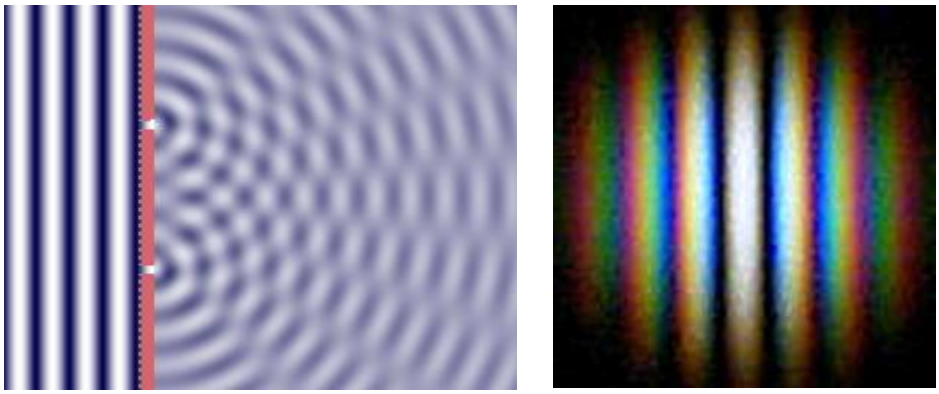


Interferencia dvoch vĺn vychádzajúcich z dvojštrbiny

Obe štrbiny budeme považovať za bodové (v reze) zdroje svetla. Je jasné, že miesto na tienidle so súradnicou *y* (pozri obrázok) je bližšie k vrchnej štrbine ako k spodnej, a preto amplitúda vlny vychádzajúcej z vrchnej štrbiny je v mieste *y* vyššia, než amplitúda vlny vychádzajúcej zo spodnej štrbiny. Pre svetlo sú však praktické vzdialenosti štrbín v ráde mikrometrov, čo je v porovnaní so všetkými rozumnými vzdialenosťami tienidla (centimetre až metre) zanedbateľná hodnota. Preto budeme amplitúdy oboch vĺn v mieste tienidla považovať za rovnaké (jednotkové). Potom pre výslednú amplitúdu v mieste *y* platí:

$$A = \exp\left(-ikr_1\right) + \exp\left(-ikr_2\right)$$
$$r_1 = \sqrt{L^2 + (y - d/2)^2}$$
$$r_2 = \sqrt{L^2 + (y + d/2)^2}$$

Nasledujúci obrázok ilustruje interferenciu vĺn na dvojštrbine pomocou Huyghensovho princípu:

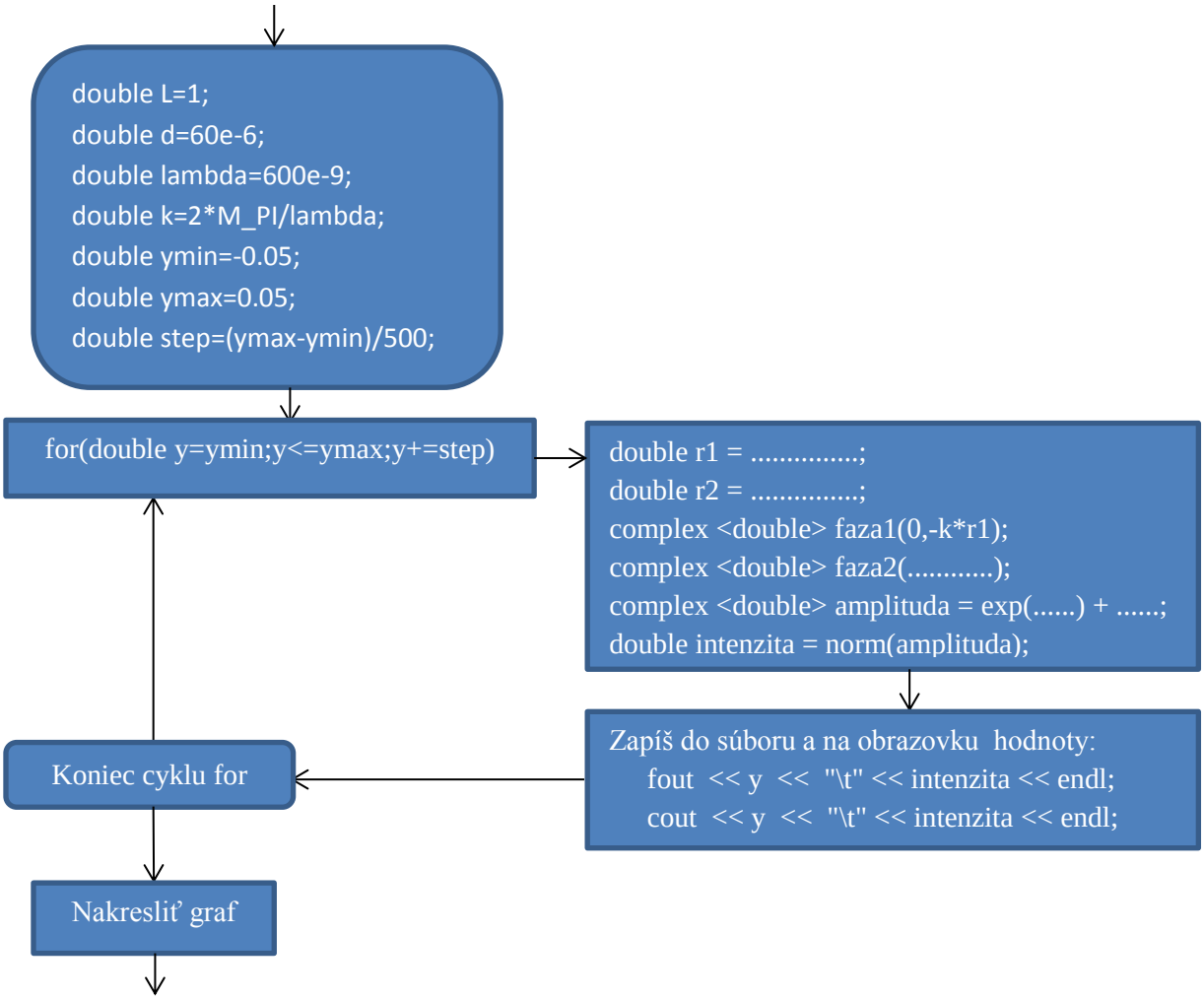


Interferencia vĺn na dvojštrbine.
Vľavo – ilustrácia Huyghendovho princípu, vpravo – fotografia interferencie na dvoch kruhových otvoroch
*Zdroj:*http://www.itp.uni-hannover.de/~zawischa/ITP/multibeam.html

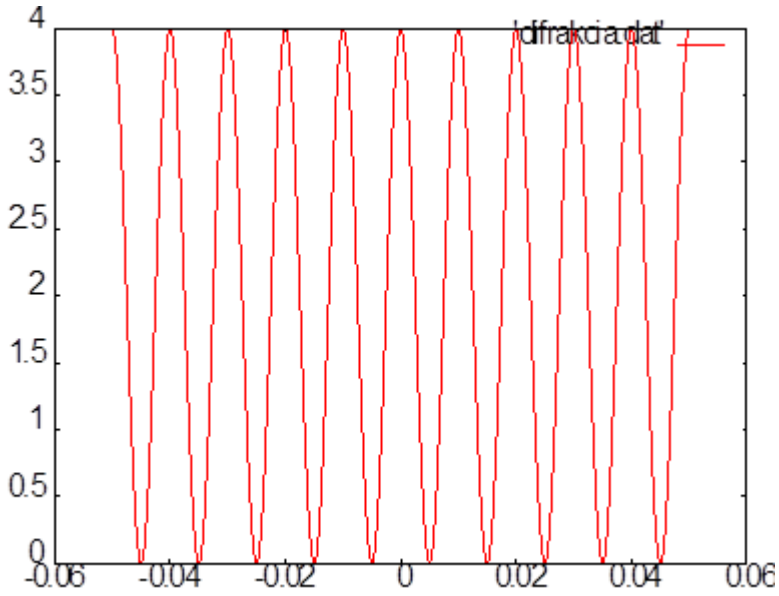
Úloha: Napíšte program, ktorý vypočíta profil jasu interferenčného obrazca na tienidle. Predpokladajte žlté svetlo λ = 600 nm, vzdialenosť štrbín d = 60 µm a vzdialenosť tienidla L = 1m. Vypočítaný profil zobrazte pomocou programu Grace alebo GNUplot.

Úloha: Preskúmajte vplyv vzdialenosti štrbín na hustotu interferenčných prúžkov.

Návod: S výhodou využite knižnicu `<complex>`, vďaka ktorej môžete ľahko sčítavať komplexné čísla, počítať hodnotu exp(x) komplexného čísla x, ako aj získať modul komplexného čísla. Vypočítajte jas na tienidle v 500 bodoch s hodnotou y = <-0,05 m ; +0,05 m>. Použite aj nasledujúci vývojový diagram jadra programu.

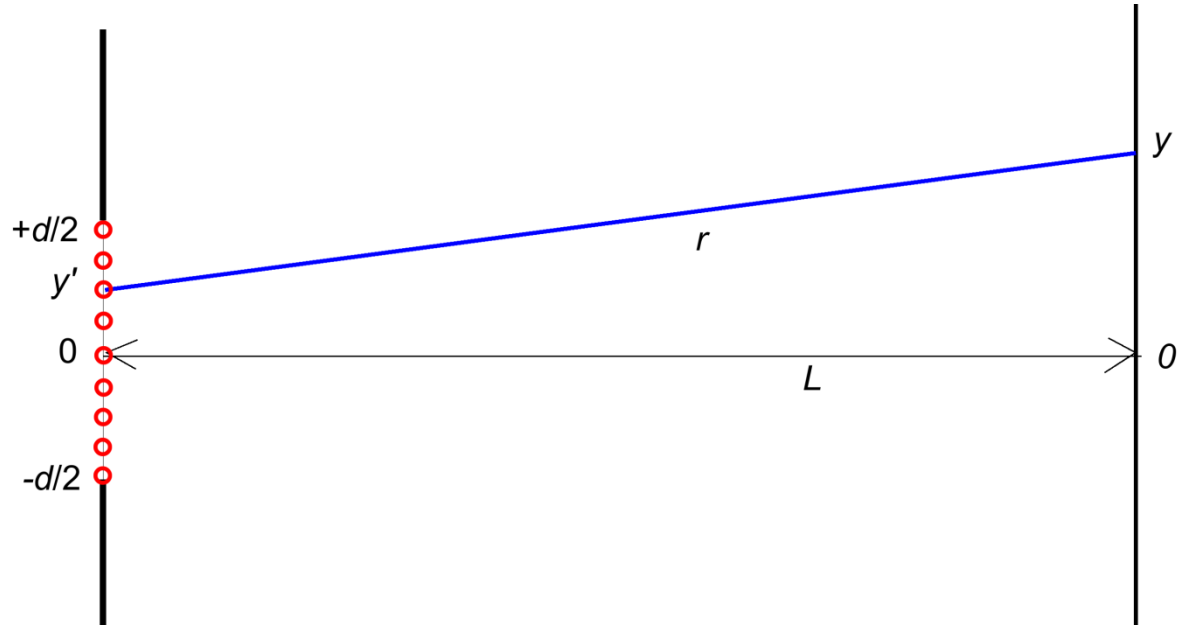


Ak máte program urobený správne, výsledkom bude profil jasu sústavy interferenčných prúžkov:

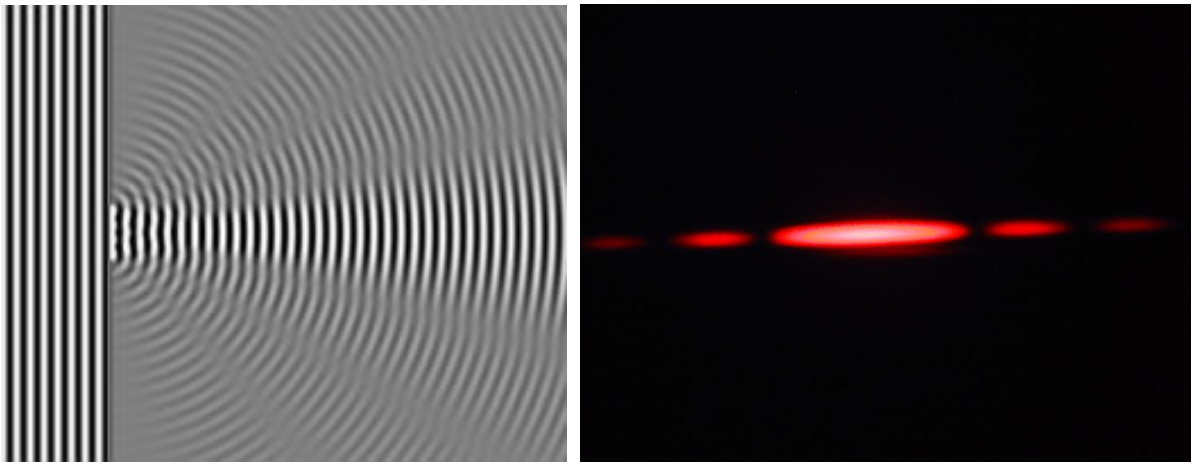


11.5. Interferencia vĺn na jednej (širšej) štrbine

Jednu širšiu štrbinu si môžeme predstaviť ako súbor navzájom sa dotýkajúcich tenkých štrbín. Nebudeme teda sčítavať iba dve vlny, ale v cykle veľké množstvo vĺn vychádzajúcich z rôznych miest štrbiny:



Výsledný interferenčný obraz vyzerá takto:



Ohyb svetla na jednej štrbine.
Vľavo – ilustrácia Huyghensovho princípu, vpravo – difrakcia lúča z laserového ukazovadla na štrbine. Zdroj: http://creationwiki.org/Optics, http://www1.union.edu/newmanj/Physics100/Light%20as%20a%20Wave/light_as_a_wave.htm

Úloha: Upravte váš program tak, že nebude sčítavať iba dve vlny, ale vlny z množstva zdrojov nachádzajúcich sa v štrbine. Zdroje v štrbine musia byť od seba vzdialené iba málo, porovnateľne s vlnovou dĺžkou svetla.

Nájdite vhodný počet zdrojov v štrbine tak, aby ich bolo čo najmenej (aby program počítal úlohu rýchlo), ale aby pritom vypočítaný obrazec bol správny.

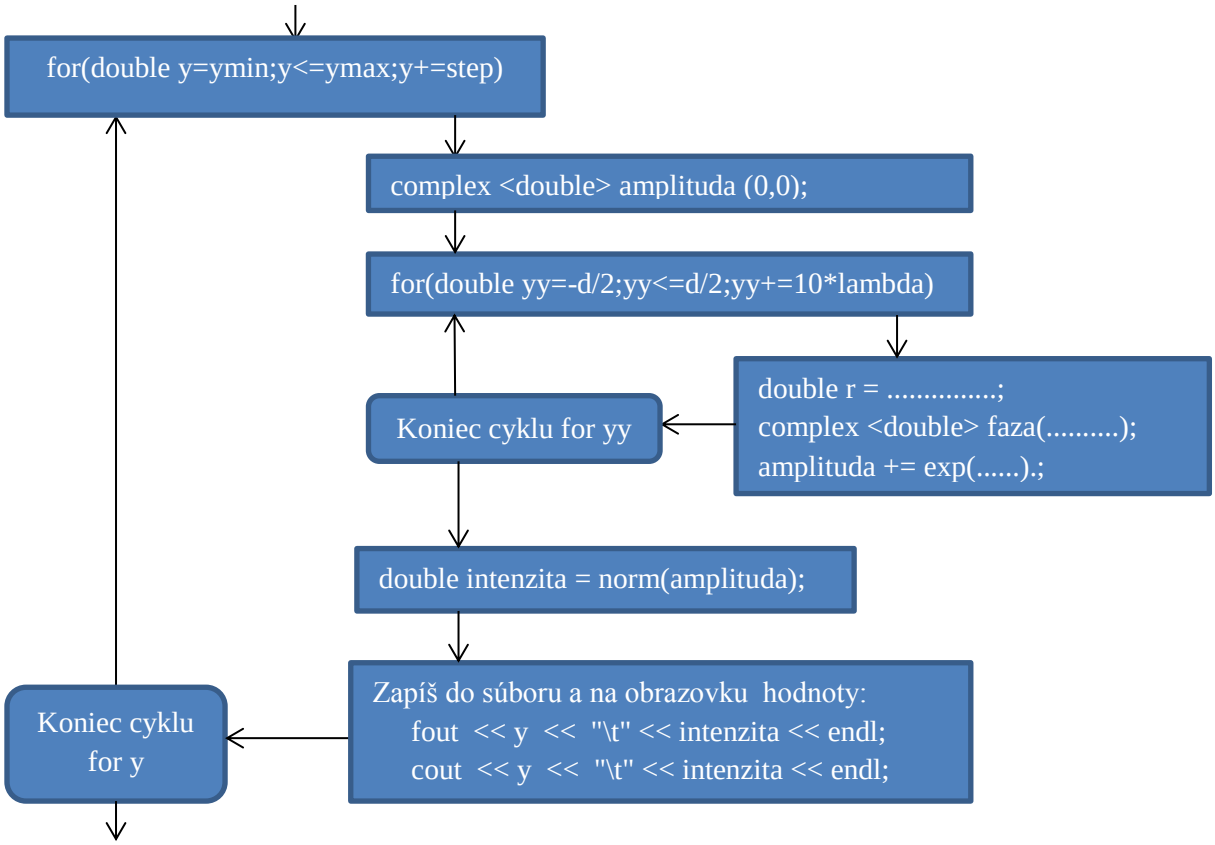
Úloha: Preskúmajte vplyv šírky štrbiny a vlnovej dĺžky na hustotu interferenčných prúžkov.

Návod: Do programu treba vložiť ďalší vnorený cyklus, kde pre každú hodnotu *y* budeme v cykle pripočítavať amplitúdy od jednotlivých zdrojov vĺn v štrbine. Vždy pred vstupom do tohto vnoreného cyklu treba ale vynulovať amplitúdu. Po sčítaní amplitúd od všetkých zdrojov v štrbine môžeme nakoniec vypočítať intenzitu svetla na tienidle pre dané *y*.

Na začiatok dajte krok vo vnorenom cykle 10 *lambda*. Výpočet nebude úplne presný, ale bude rýchly.

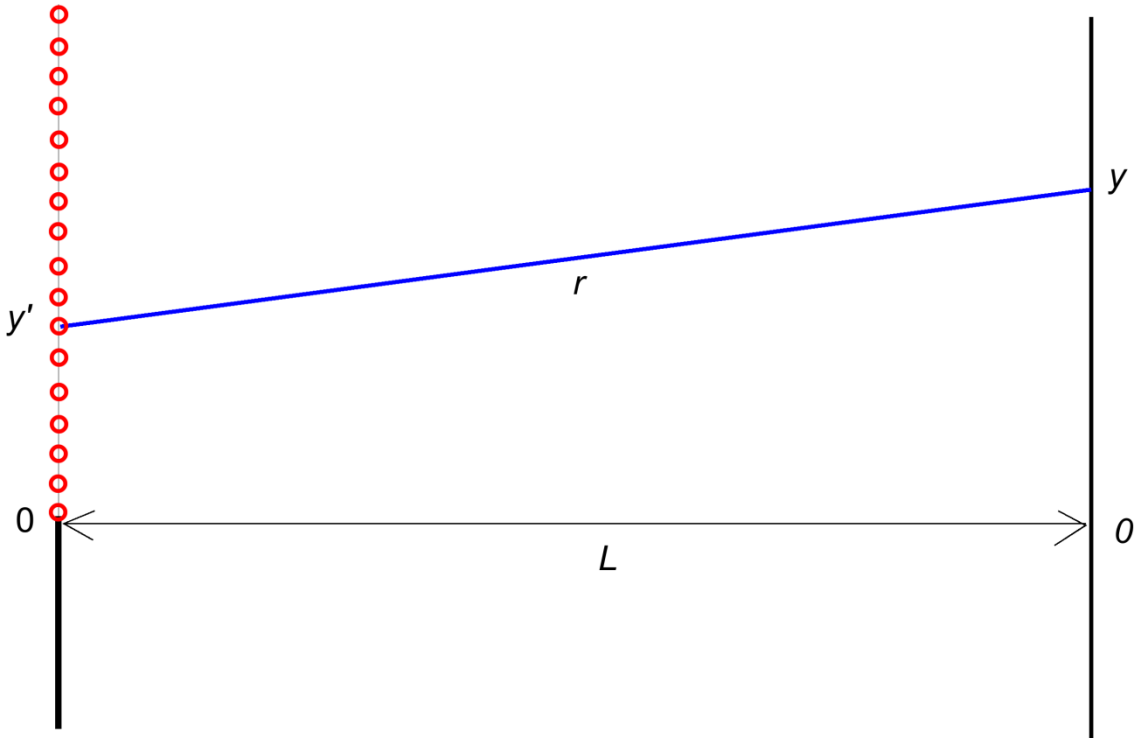
Na posúdenie presnosti výpočtu si všímajte polohu miním. Pre zadané parametre systému by mali byť vo vzdialenosti 0,010 m - 0,020 m – 0,030 m - ...

Použite aj nasledujúci vývojový diagram upraveného jadra programu. Ujasnite si, prečo bolo treba urobiť práve takéto úpravy.

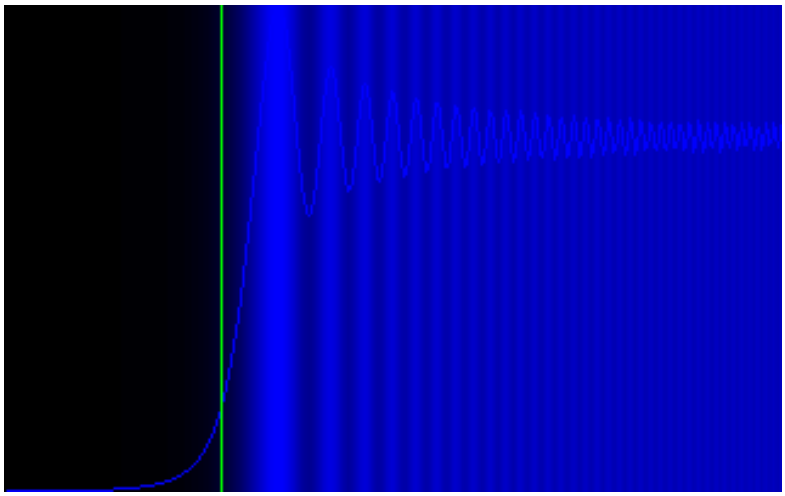


11.6. Ohyb svetla na hrane

Ohybové javy možno pozorovať aj na hrane. Vtedy je polovica vlnoplochy zakrytá tienidlom a druhú polovicu musíme nahradiť veľkým (nekonečným) počtom zdrojov vĺn:



Interferenčný obrazec vyzerá takto:



Ohyb svetla na hrane.
Zdroj: http://laserstars.org/glossary/diffract/plots/

Úloha: Upravte váš program tak, že bude počítať ohyb svetla na hrane. Nájdite vhodnú hustotu zdrojov vĺn tak, aby výpočet bol dostatočne presný. Nájdite vhodnú hodnotu "nekonečna" tak, aby výpočet bol dostatočne rýchly, ale pritom aj presný.

Návod:

1. Interferenčné prúžky budú hustejšie, upravte preto hodnoty *ymin* = - 0,005 a *ymax* = 0,005.

2. Upravte hranice vnoreného cyklu (podľa premennej *yy*) tak, že začína pri *yy* = 0 a končí pri veľkom násobku hodnoty *ymax* (náhrada nekonečna). Experimentovanie začnite s koncom cyklu pri hodnote *yy* = 5 *ymax*.a postupne hranicu cyklu zvyšujte.

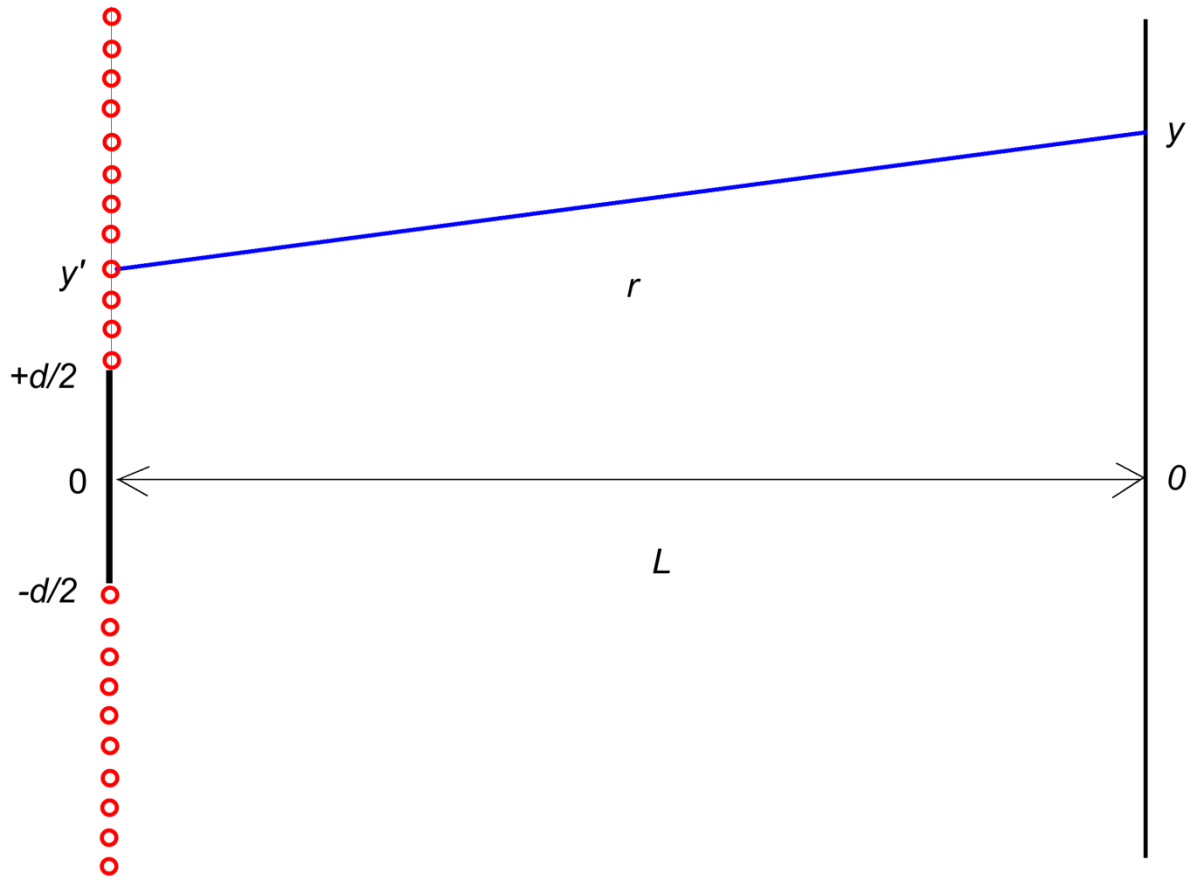
3. Nájdite aj vhodný prírastok premennej *yy* v cykle. Experimentovanie začnite s krokom 10 *lambda*. Výpočet síce nebude celkom presný, ale bude rýchly.

Všimnite si, že na tienidle nie je polovičné osvetlenie presne oproti hrane.

Úloha: Aká intenzita svetla (v porovnaní s priamo osvetlenou časťou) je na úrovni hrany?

11.7. Domáca úloha - ohyb svetla na drôtiku

Ohybové javy možno pozorovať aj na drôtiku. Vtedy je zakrytá časť vlnoplochy, zvyšok (na obe strany od drôtika) musíme nahradiť veľkým (nekonečným) počtom zdrojov vĺn:



Úloha: Upravte váš program tak, že bude počítať ohyb svetla na drôtiku s hrúbkou *d*. Nájdite vhodnú hustotu zdrojov vĺn tak, aby výpočet bol dostatočne presný. Nájdite vhodnú hodnotu "nekonečna" tak, aby výpočet bol dostatočne rýchly, ale pritom aj presný.

Návod:

Experimentovať môžete začať s rovnakými parametrami, ako v predchádzajúcej úlohe (nekonečná nahradiť hodnotami ± 5 *ymax* a hustotu zdrojov zvoliť 10 *lambda*). Výpočet nebude síce veľmi presný, ale bude rýchly.

Mali by ste vypočítať profil nasledujúceho interferenčného obrazca:

Ohyb svetla na tenkom drôtiku
Zdroj: http://www.physics.montana.edu/demonstrations/apparatus/6_optics/demos/thinwirediffraction.html

Viete vysvetliť, prečo v mieste presne oproti stredu drôtika nie je osvetlenie minimálne?

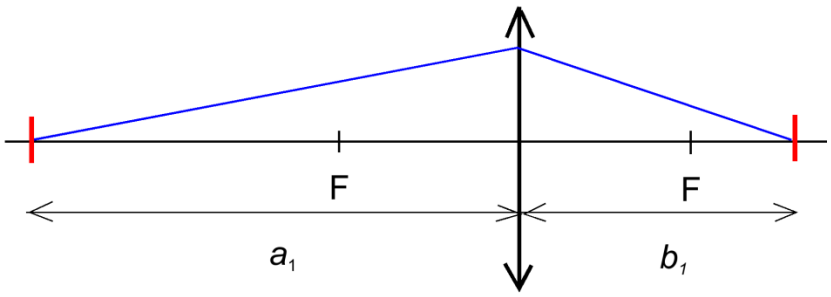
12. Sústavy šošoviek, hlavné roviny, zobrazovacia rovnica

12.1. Vedomosti potrebné na absolvovanie cvičenia

Na tomto cvičení vystačíte s vedomosťami, ktoré ste získali na predchádzajúcich cvičeniach.

12.2. Zobrazovacia rovnica tenkej spojnej šošovky

Vo vzdialenosti a_1 od tenkej spojnej šošovky s ohniskovou vzdialenosťou f_1 = 6 cm sa nachádza predmet, ktorý šošovka premieta na tienidlo. Tienidlo je vo vzdialenosti b_1 od šošovky (obrázok 1).



Obrázok 1. Tenká spojná šošovka

V stave zaostrenia je splnená zobrazovacia rovnica šošovky

$$\frac{1}{a_1}+\frac{1}{b_1}=\frac{1}{f_1}$$

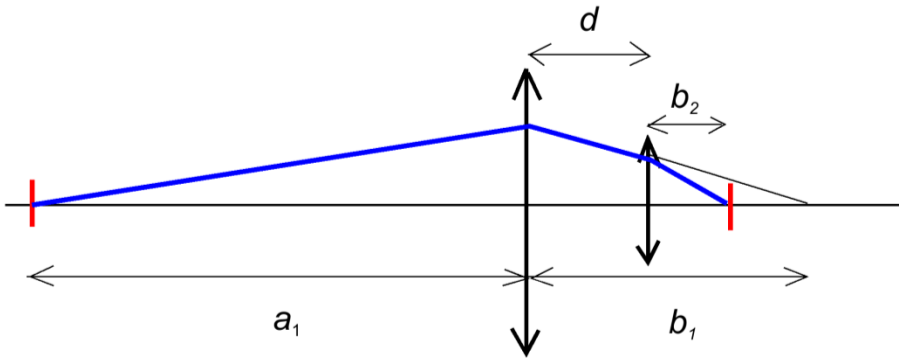
Pre zväčšenie obrazu platí vzťah

$$Z_1=-\frac{b_1}{a_1}$$

Úloha: Napíšte program, do ktorého zadáte z klávesnice číslo a_1 a program vypíše hodnotu čísla b_1 a taktiež vypíše zväčšenie obrazu Z_1. Overte funkčnosť programu tak, že zadáte do programu vzdialenosť obrazu od šošovky, ktorá je väčšia, ale aj menšia, než ohnisková vzdialenosť šošovky f_1. Uvedomte si, že vzdialenosť môže vyjsť aj záporná (zdanlivý obraz), rovnako môže vyjsť aj záporné zväčšenie (prevrátený obraz).

12.3. Sústava dvoch tenkých spojných šošoviek

Na obrázku 2 je znázornená sústava šošoviek, v ktorej sme k prvej šošovke pridali vo vzdialenosti d = 2 cm druhú šošovku, ktorej ohnisková vzdialenosť je f_2 = 4 cm.



Obrázok 2. Sústava dvoch tenkých spojných šošoviek

Prvá šošovka vytvára pre druhú šošovku zdanlivý obraz vo vzdialenosti

$$a_2 = d - b_1$$

od druhej šošovky. Všimnite si, že a_2 je záporné číslo.

Výsledné zväčšenie sústavy nájdeme podľa vzťahu

$$Z = Z_1 \cdot Z_2$$

kde Z_2 je zväčšenie obrazu druhou šošovkou.

Úloha: Doplňte program tak, že keď doň z klávesnice zadáte z číslo a_1, program vypíše hodnotu čísla b_2 a taktiež vypíše výsledné zväčšenie obrazu $Z = Z_1.Z_2$. Overte funkčnosť programu tak, že zadáte do programu vzdialenosť predmetu od prvej šošovky rovnakú, ako je ohnisková vzdialenosť prvej šošovky (a_1 = 6 cm), čím prvú šošovku opustia rovnobežné lúče. Tieto druhá šošovka sfokusuje do svojho ohniska, preto b_2 = 4 cm.

12.4. Hlavné roviny optickej sústavy

V každej optickej sústave hrajú dôležitú úlohu tzv. *hlavné roviny*. Sú to také roviny, že ak položíme predmet do jednej z nich, premietne sa do druhej so zväčšením Z = 1. U tenkej šošovky obe roviny splývajú do jednej, ktorá prechádza stredom šošovky (predmet a obraz sú totožné). Upozornenie: hlavné roviny sa môžu nachádzať aj vo vnútri optickej sústavy (medzi šošovkami).

Úloha: Skusmo (hľadaním) nájdite polohu hlavných rovín optickej sústavy zloženej z našich dvoch tenkých spojných šošoviek, t.j. nájdite vzdialenosti h_1, h_2 hlavných rovín od prvej a druhej šošovky.

12.5. Ohnisková vzdialenosť optickej sústavy

Ohniskové roviny optickej sústavy sú také roviny, že predmet umiestnený v nich sa zobrazí v nekonečne ($b_2 = \infty$) a s nekonečným zväčšením ($Z = \infty$). Alternatívou je umiestniť predmet vo veľmi veľkej vzdialenosti ($a_1 = \infty$), vtedy sa predmet zobrazí do ohniskovej roviny.

Úloha: Skusmo (hľadaním) nájdite polohu ohniskových rovín optickej sústavy zloženej z našich dvoch tenkých spojných šošoviek, t.j. nájdite vzdialenosti o_1, o_2 ohniskových rovín od prvej a druhej šošovky.

12.6. Zobrazovacia rovnica optickej sústavy

Pre optickú sústavu platí analogická zobrazovacia rovnica, ako pre tenkú šošovku, akurát všetky vzdialenosti sa merajú od hlavých rovín:

$$\frac{1}{l_1} + \frac{1}{l_2} = \frac{1}{f}$$

kde

$$l_1 = a_1 - h_1$$

$$l_2 = a_2 - h_2$$

$$f = o_1 - h_1 = o_2 - h_2$$

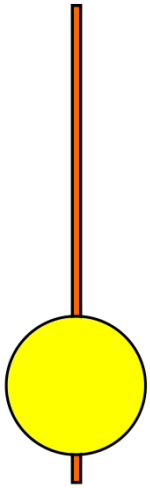
Úloha: Overte zobrazovaciu rovnicu sústavy

$$\frac{1}{l_2} = \frac{1}{f} - \frac{1}{l_1}$$

tak, že nakreslíte graf závislosti $y = 1/(a_2 - h_2)$ od $x = 1/(a_1 - h_1)$ v rozsahu hodnôt $a_1 = 0$ cm až 10 000 cm s krokom 100 cm. Ak rovnica platí, mali by ste dostať priamku. Z priesečníku priamky s osou y nájdite ohniskovú vzdialenosť sústavy a porovnajte ju s vypočítanou.

13. Otestujte sa. Hodinové kyvadlo.

13.1. Úvod



Na tejto úlohe, ktorej obťažnosť zodpovedá tomu, čo sa od vás bude očakávať na skúške, si otestujte, či ste zvládli základy programovania v jazyku C++.

Cieľom je oboznámiť sa s periódou kmitov kyvadla, a to aj pri veľkých výchylkách.

Kyvadlo pozostáva z drevenej tyče dĺžky l = 0,9 m s hmotnosťou m = 0,07 kg, ktorá sa môže voľne otáčať okolo horného konca (obrázok). Pri jej spodnom konci je zavesené závažie v tvare mosadzného disku s polomerom R = 0,1 m a hmotnosťou M = 0,3 kg, pričom *stred závažia* je vo vzdialenosti x = 12,4 cm od spodného konca tyče. Závažie možno po tyči jemne posúvať hore a dolu.

Moment zotrvačnosti kyvadla okolo bodu otáčania je

$$I = \frac{1}{3}ml^2 + M(l-x)^2 + \frac{1}{2}MR^2$$

Perióda kmitov takéhoto kyvadla pri veľmi malých výchylkách kyvadla je daná vzťahom

$$T = 2\pi\sqrt{\frac{I}{m_c g h}}$$

kde m_c je celková hmotnosť kyvadla

$$m_c = m + M$$

a hodnota h je vzdialenosť ťažiska od bodu závesu:

$$h = \frac{\frac{ml}{2} + M(l-x)}{m+M}$$

Hodnotu gravitačného zrýchlenia predpokladajte g = 9,80665 m/s^2.

13.2. Perióda kmitov kyvadla pre malé výchylky

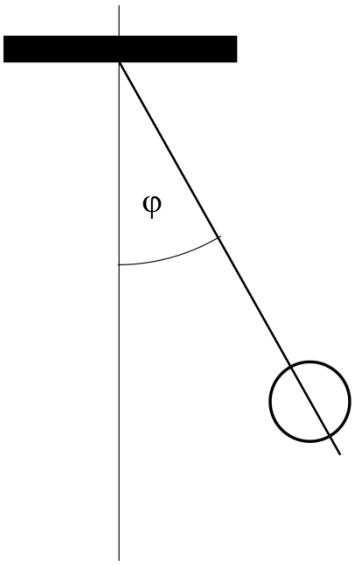
Napíšte program, ktorý vypočíta periódu kmitov kyvadla (v sekundách) pri veľmi malých výchylkách. Všetkým premenným odporúčame dať typ `double`. Hodnoty parametrov kyvadla nezadávajte z klávesnice, stačí ich mať vložené priamo do programu. Koľko kmitov urobí kyvadlo za 24 hodín = 86400 s?

13.3. Nastavovanie periódy kmitov

Zistite, o koľko sa zmení perióda kmitov kyvadla, ak závažie zdvihneme (posunieme nahor) o 1 mm. O koľko kmitov viac urobí takto skrátené kyvadlo za 24 hodín = 86400 s? O koľko sekúnd nadbehnú hodiny so skráteným kyvadlom za 24 hodín?

13.4. Pohyb kyvadla pri veľkých výchylkách

Vzorec pre periódu kyvadla, uvedený v úvode, platí iba pre veľmi malé výchylky kyvadla. Pre väčšie výchylky už pohyb kyvadla nie je sínusový, ale závislosť výchylky kyvadla od času je komplikovanejšia.



Pre uhol vychýlenia φ, uhlovú rýchlosť ω a uhlové zrýchlenie ε platia vzťahy:

$$\varepsilon = -\frac{m_c g h}{I}\sin\varphi$$

$$\frac{d\omega}{dt} = \varepsilon$$

$$\frac{d\varphi}{dt} = \omega$$

Napíšte program, ktorý vypočíta závislosť výchylky kyvadla od času. Na začiatku predpokladajte:

$$\varphi(t = 0) = \varphi_0$$

$$\omega(t = 0) = 0$$

čiže kyvadlo má maximálnu výchylku a stojí (uhlová rýchlosť je nulová).

Úlohu riešte Eulerovou metódou v diskrétnych časových okamihoch t_i s krokom Δt. Vtedy sa diferenciálne rovnice zmenia na diferenčné:

$$\varepsilon_i = -\frac{m_c g h}{I}\sin\varphi_{i-1}$$

$$\omega_i = \omega_{i-1} + \varepsilon_i \Delta t$$

$$\varphi_i = \varphi_{i-1} + \omega_i \Delta t$$

Výpočet nechajte bežať v časovom intervale t_i = 0s ... 5s pre počiatočnú výchylku φ_0 = 0,1 rad, vypisujte do obrazovky aktuálnu hodnotu času a výchylky a overte, že riešenie zodpovedá realite (uhol sa periodicky mení).

13.5. Perióda kyvadla pri veľkých výchylkách

Program z úlohy 4 upravte tak, že výpočet skončí okamžite pri zmene znamienka uhlovej rýchlosti, čo zodpovedá okamihu najväčšej výchylky kyvadla na opačnej strane, než začínalo svoj pohyb. Vtedy kyvadlo vykonalo práve polovicu periódy svojho pohybu. Vypíšte takto určenú periódu kmitov kyvadla na obrazovku. Časový krok zvoľte taký malý, aby ste periódu určili s presnosťou 0,0001s. Pred výpočtom vráťte kyvadlo na pôvodnú výšku x = 0,124 m.

Aká je perióda kmitov kyvadla, ak počiatočná výchylka je 90^o?

Aká malá musí byť výchylka, aby sa perióda kmitov líšila od hodnoty pre malé kmity o menej než 0,0001s?

13.6. Závislosť periódy kyvadla od amplitúdy kmitov

Nakreslite graf závislosti periódy kmitov kyvadla od amplitúdy kmitov. Amplitúdu meňte od 5^o po 90^o s krokom 5^o. Pre nulovú amplitúdu kmitov výpočet nerobte.

Návod:

Program z úlohy 5 doplňte o vonkajší cyklus, v ktorom budete počiatočnú hodnotu φ_0 meniť od 5^o po 90^o. Pre každú hodnotu φ_0 spustite výpočet periódy T a po skončení uložte do súboru dvojicu hodnôt φ_0 a T. Po skončení vonkajšieho cyklu budete mať v súbore uložené dvojice hodnôt pre všetky amplitúdy kmitov, ktoré môžete priamo vykresliť.

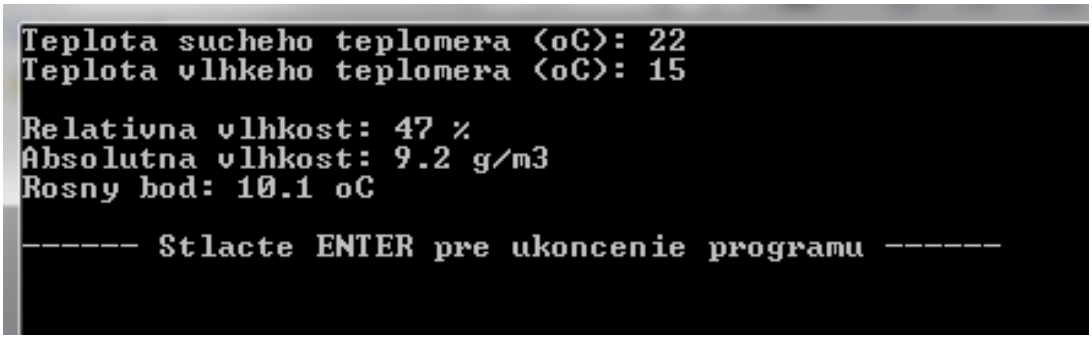
Krok Δt metódy výrazne predĺžte, aby výpočet netrval dlho. Presnosť 0,001 s je pre nakreslenie grafu dostatočná.

14. Výsledky úloh

Na záver uvádzame výsledky úloh (výstupy programov). Majú slúžiť na to, aby ste si mohli overiť, že vami napísaný program je správny. Neuvádzame však samotný kód programov, lebo tú istú vec možno naprogramovať viacerými spôsobmi a taktiež by prezradenie kódu výrazne oslabilo tréning v písaní a ladení programov. Z rovnakých dôvodov neuvádzame ani výsledky domácich úloh.

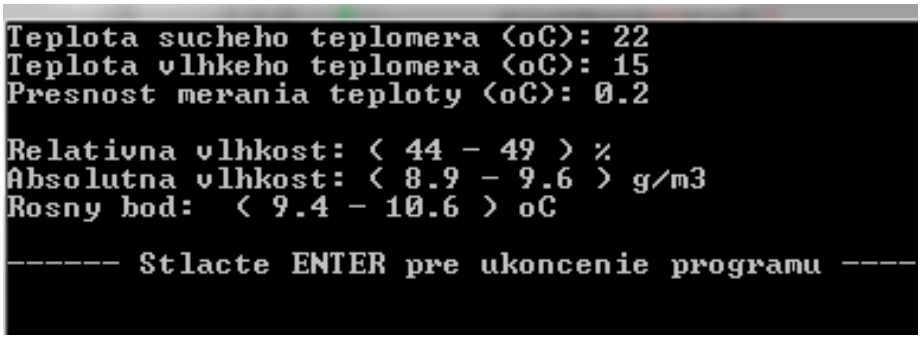
2.4. Výpočet relatívnej vlhkosti pomocou Asmanovho psychrometra

Príklad výstupu programu:



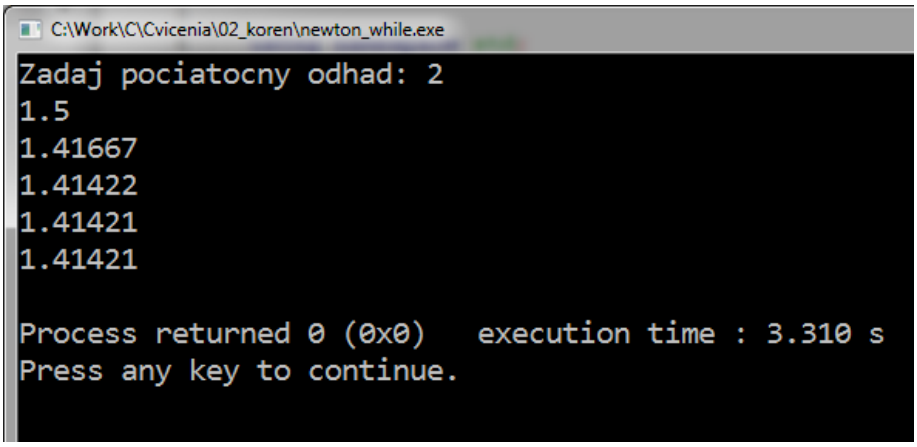
2.5. Domáca úloha – zohľadnenie presnosti merania teploty

Príklad výstupu programu:



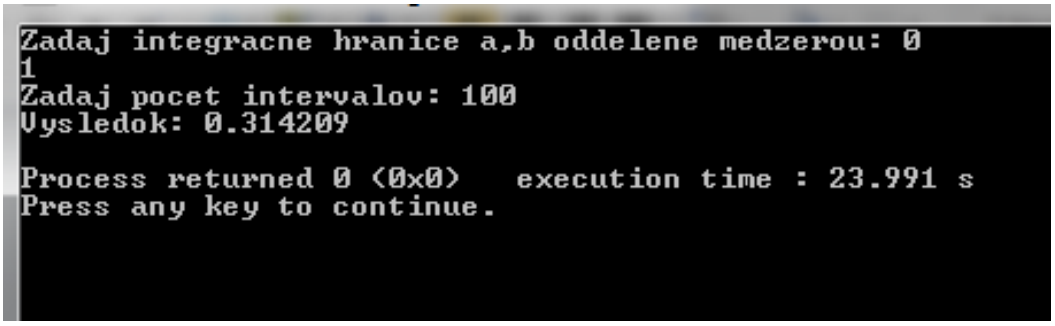
3.3. Hľadanie koreňa funkcie Newtonovou metódou

Výstup programu:



4.2. Obdĺžniková metóda integrovania

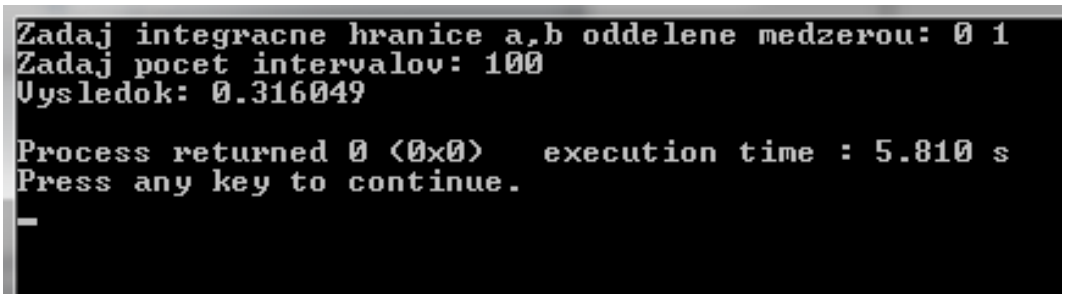
Príklad výstupu programu:



Požadovanú presnosť dosiahneme pri počte intervalov n= 250 000.

4.3. Lichobežníková metóda integrovania

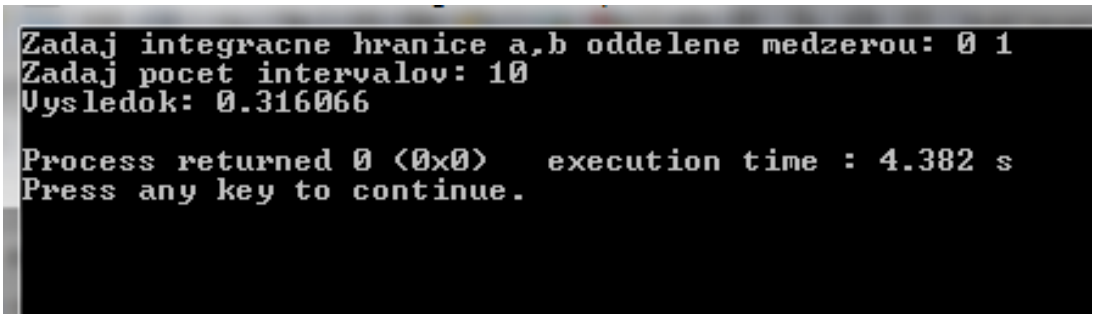
Príklad výstupu programu:



Požadovanú presnosť dosiahneme pri počte intervalov 390, čo je takmer 1000-krát menej výpočtov, než pri použití obdĺžnikovej metódy.

4.4. Parabolická (Simpsonova) metóda integrovania

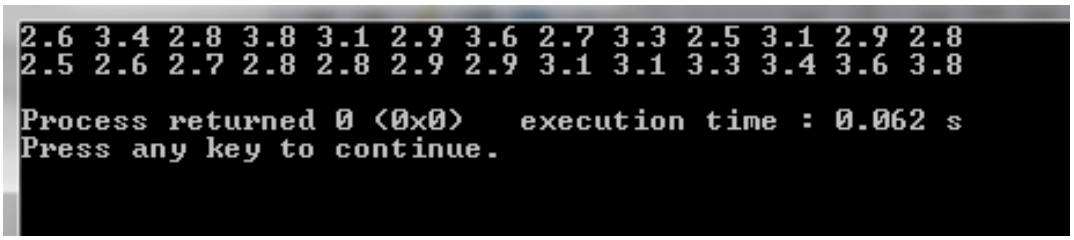
Príklad výstupu programu:



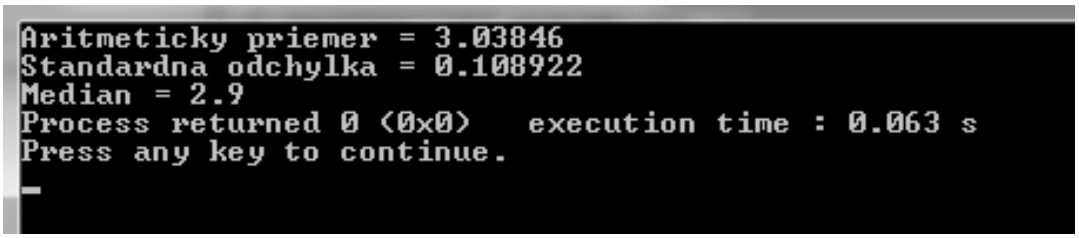
Požadovanú presnosť dosiahneme pri počte intervalov 24, čo je takmer 10-krát menej výpočtov, než pri použití lichobežníkovej metódy. Dá sa pekne vidieť, že pri použití nepárneho počtu intervalov je výsledok výrazne nepresnejší.

5.3. Medián ako vhodnejší odhad najpravdepodobnejšej hodnoty pri vybočujúcich údajoch

Triedenie:



Štatistika:

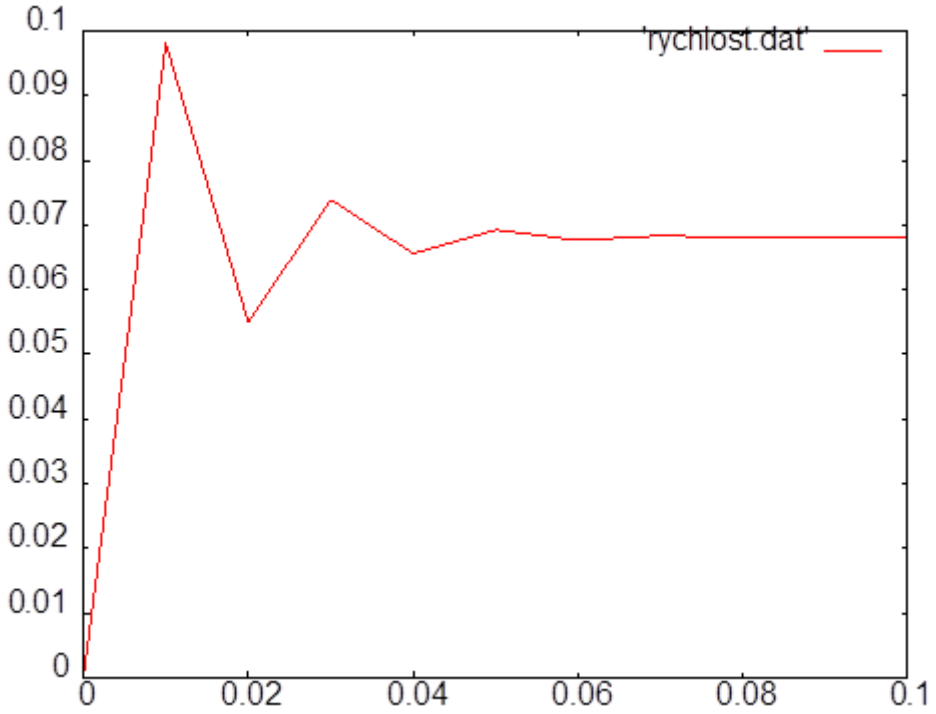


6.3. Riešenie sústavy lineárnych algebraických rovníc pomocou inverzie

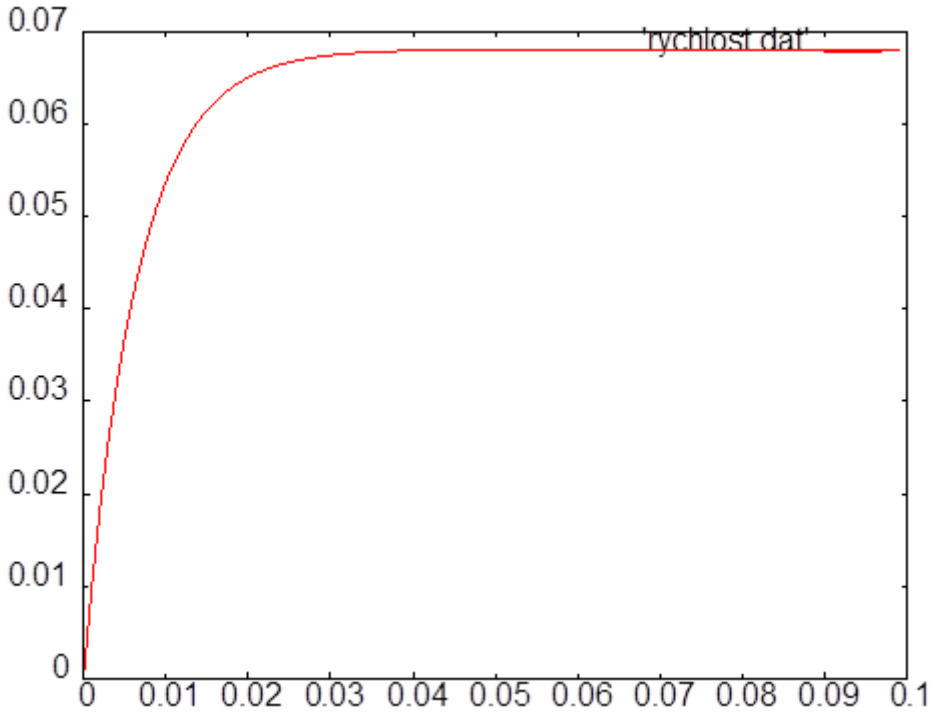
```
Pocet rovnic: 3
Zadaj rovnicu 0: 1 2 3 4
Zadaj rovnicu 1: 2 3 4 5
Zadaj rovnicu 2: 3 4 5 6
x[0] = -1
x[1] = -2
x[2] = 0.5

Process returned 0 (0x0)   execution time : 17.975 s
Press any key to continue.
```

7.4. Eulerova metóda riešenia diferenciálnych rovníc s počiatočnou podmienkou

Po spustení výpočtu (s odporúčanými parametrami) dostávame nasledujúci graf:

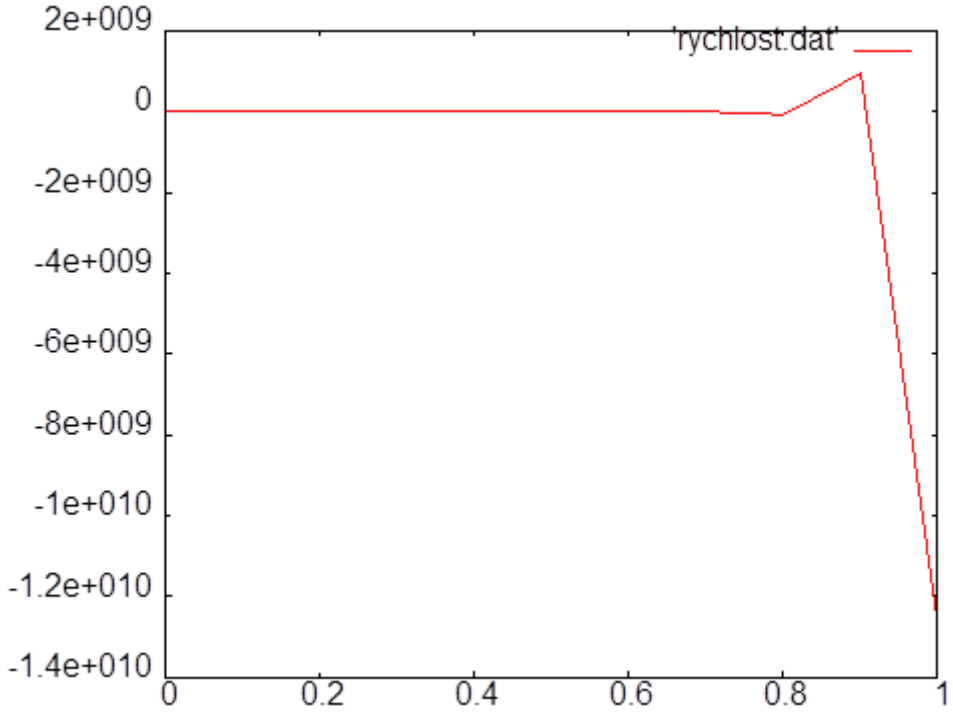


Vidíme, že krok 0,01 s je príliš hrubý. Preto zvolíme nový krok 0,001 s:

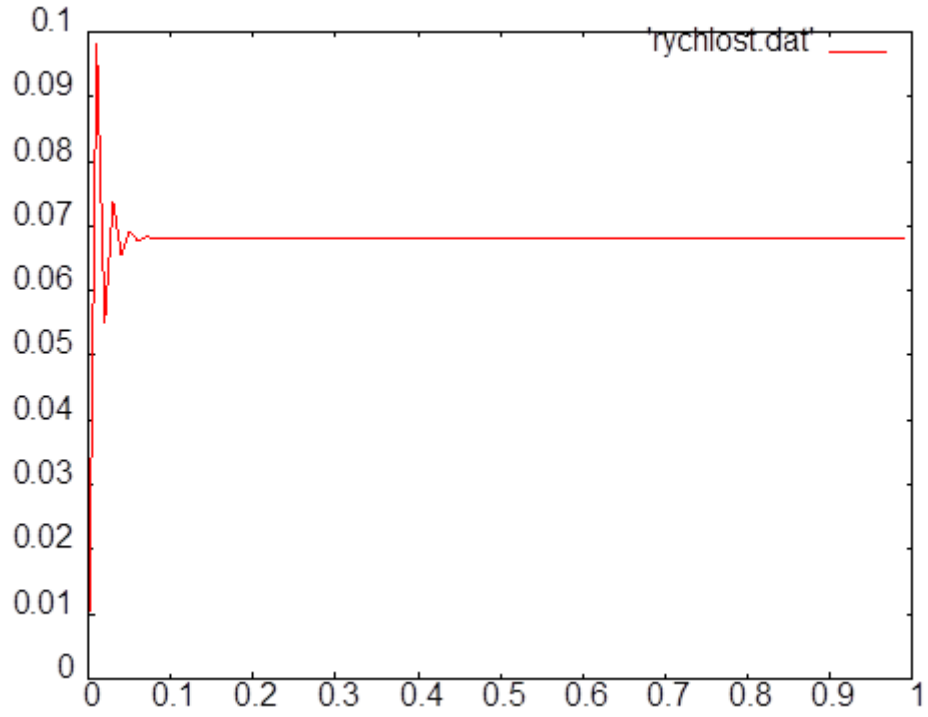


Vidíme, že k ustáleniu rýchlosti dochádza veľmi rýchlo (asi 0,05 s), ustálená rýchlosť pádu guľôčky je asi 6,8 cm/s.

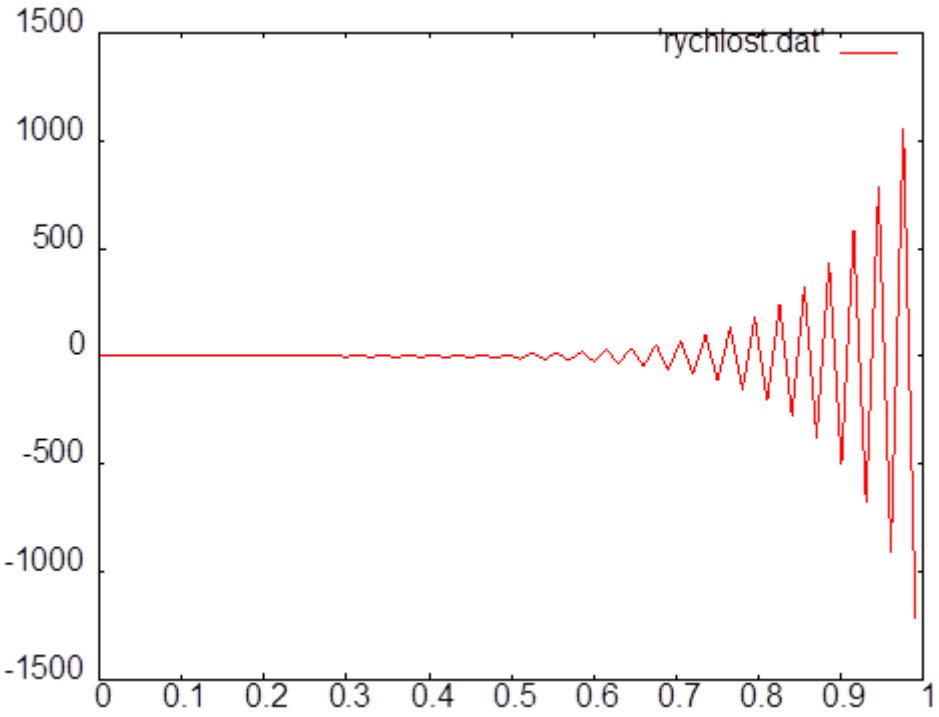
Ak zvolíme krok príliš hrubý (0,1 s) a necháme počítať 1 s pádu, dostávame:



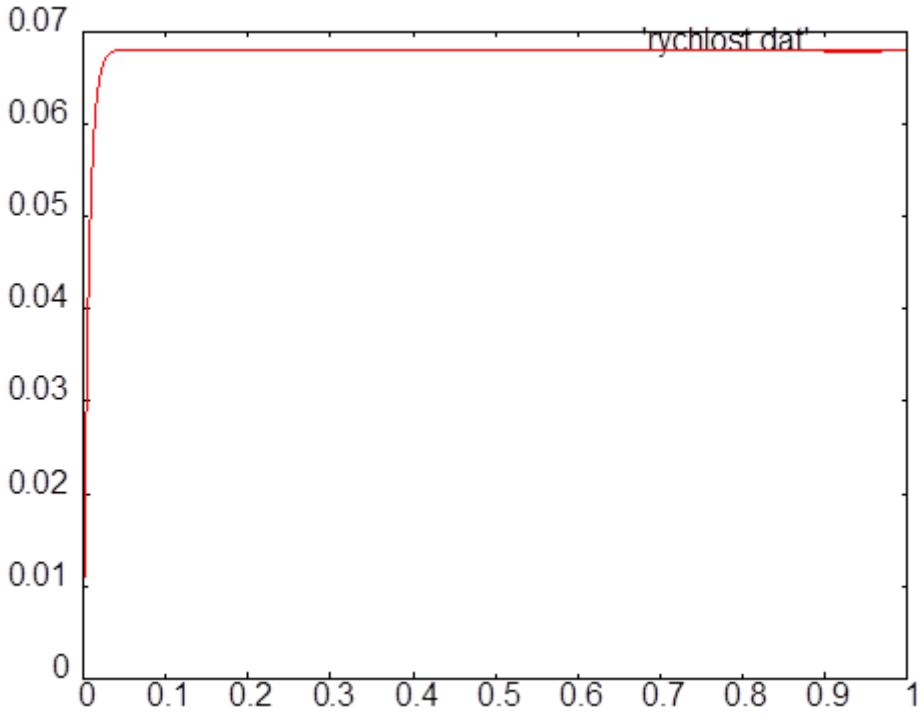
Algoritmus zlyhal – dostávame nezmyselne vysoké (a navyše záporné) hodnoty rýchlosti. Ak skúsime jemne meniť krok, zistíme, že pri 0,01 s dostávame na začiatku nerealistický "rozkmitaný" priebeh rýchlosti, ktorý sa ale aspoň blíži k realistickej výslednej rýchlosti:



Ak zvolíme krok ešte dlhší, napríklad 0,015 s, algoritmus nekonverguje, ale postupne sa "rozkmitá":



Vidíme, že nestačí mať iba dobré rovnice, ale treba aj vhodne zvoliť krok metódy, aby sme výsledku mohli veriť. Krok 0,001 s vedie síce k rozsiahlejšiemu množstvu uložených dát, ale výsledok je správny:



8.3. Generovanie náhodných čísel, štandardná odchýlka, interval 95%

Výpočet aritmetického priemeru a štandardnej odchýlky:
Postupne zvyšujeme počet čísel a zistíme, že pre viac než 1000 náhodne vygenerovaných čísel je už vždy podmienka splnená. Pre 1000 čísel dostávame:

```
Priemer = 1.03882
Std. odchylka = 2.0315

Process returned 0 (0x0)   execution time : 0.079 s
Press any key to continue.
```

Výsledok (a teda aj počet potrebných čísel), závisí samozrejme od toho, aké náhodné čísla sa vygenerovali. Uvedený výsledok je teda iba orientačný.

Výpočet intervalu 95%:

Opäť postupne zvyšujeme počet čísel a zisťujeme, kedy už podmienka ostáva splnená. Pre 5000 údajov dostávame:

```
Priemer = 0.945817
Std. odchylka = 2.01938
95% cisel je medzi <-2.95558 , 4.90417 >

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
```

Vidíme, že počet potrebných čísel je už vyšší, lebo mimo intervalu 95% sa nachádza podstatne menej čísel, než mimo intervalu určenom štandardnou odchýlkou.

Výsledok (a teda aj počet potrebných čísel), závisí samozrejme od toho, aké náhodné čísla sa vygenerovali. Uvedený výsledok je teda iba orientačný. Ďalšie výpočty budeme robiť s počtom čísel 10000 (aby sme mali rezervu).

8.4. Neistota nepriamo meranej veličiny $y = e^x$

Pre 10000 vygenerovaných čísel pri chybe merania 0,18 dostávame:

```
Priemer = 3.39716
Std. odchylka = 0.615839
95% cisel je medzi <2.34958 , 4.74748 >

Process returned 0 (0x0)   execution time : 0.094 s
Press any key to continue.
```

Vidíme, že oproti hodnote získanej linearízáciou sú všetky vypočítané hodnoty posunuté smerom k vyšším hodnotám. Dôvodom je, že na intervale danom chybou merania už exponenciálu nemôžeme aproximovať priamkou.

Po zvýšení presnosti na 0,018 dostávame:

```
Priemer = 3.35295
Std. odchylka = 0.0602868
95% cisel je medzi <3.23628 , 3.47211 >

Process returned 0 (0x0)   execution time : 0.070 s
Press any key to continue.
```

Vidíme, že zhoda s hodnotami získanými linearizáciou je výborná, exponenciálu už môžeme nahradiť (na intervale danom presnosťou merania) priamkou.

Metóda Monte Carlo dáva v porovnaní s linearizáciou spoľahlivejšie výsledky, vyžaduje však použitie počítača.

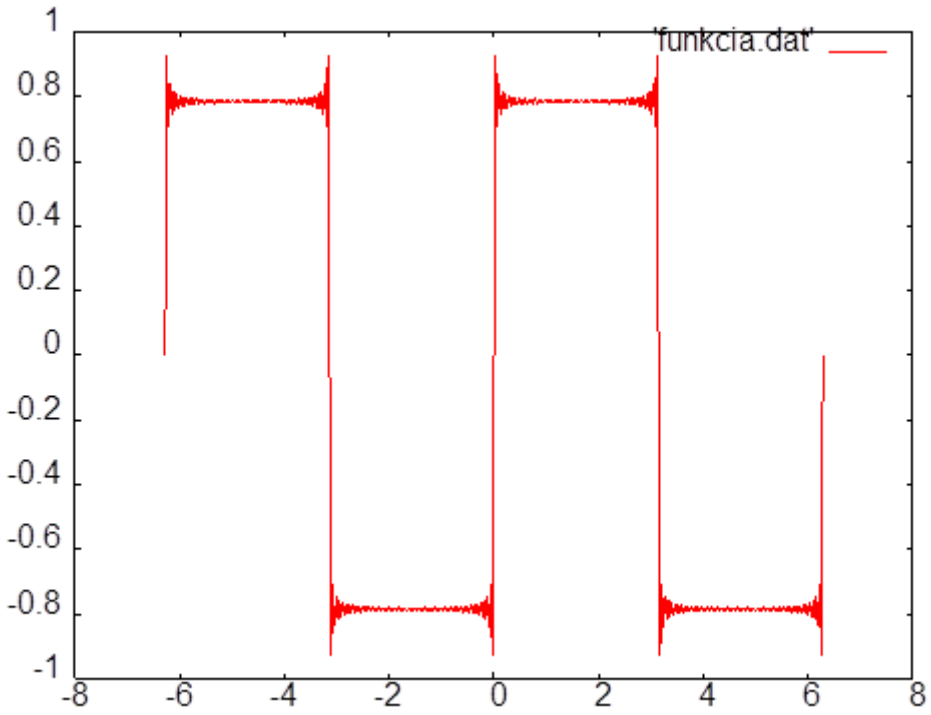
8.5. Neistota nepriamo meranej veličiny $y = a.b^2/c^3$

Použitím odporúčaných hodnôt a 10000 vygenerovaných čísel dostávame:

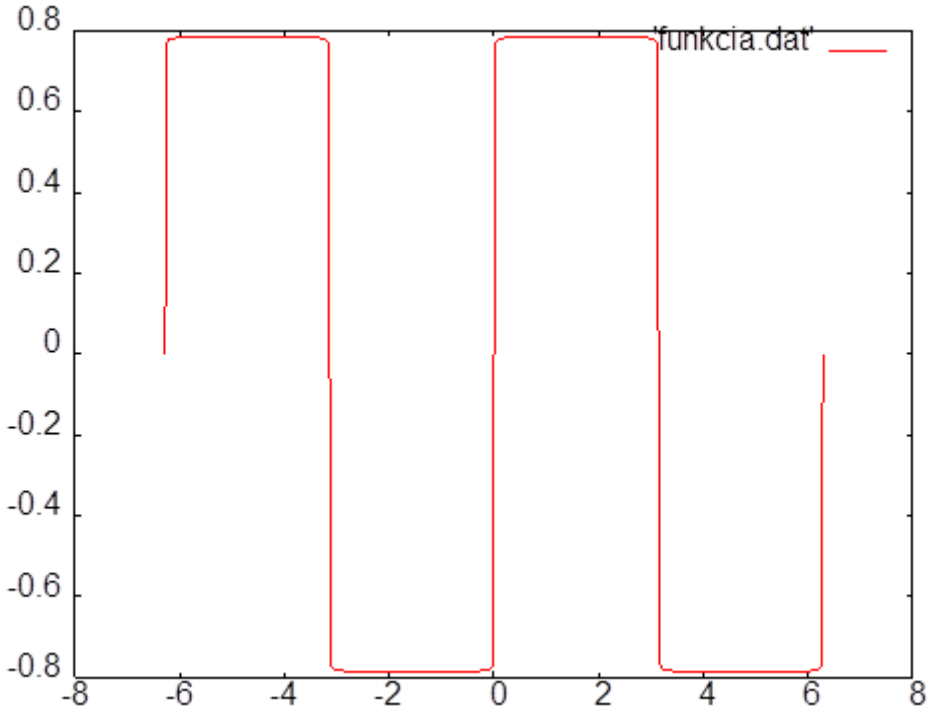
```
Priemer = 0.139056
Std. odchylka = 0.00897862
95% cisel je medzi <0.121831 , 0.15724 >

Process returned 0 (0x0)   execution time : 0.119 s
Press any key to continue.
```

9.3. Fourierove rady – všeobecný tvar. Príklady funkcií.

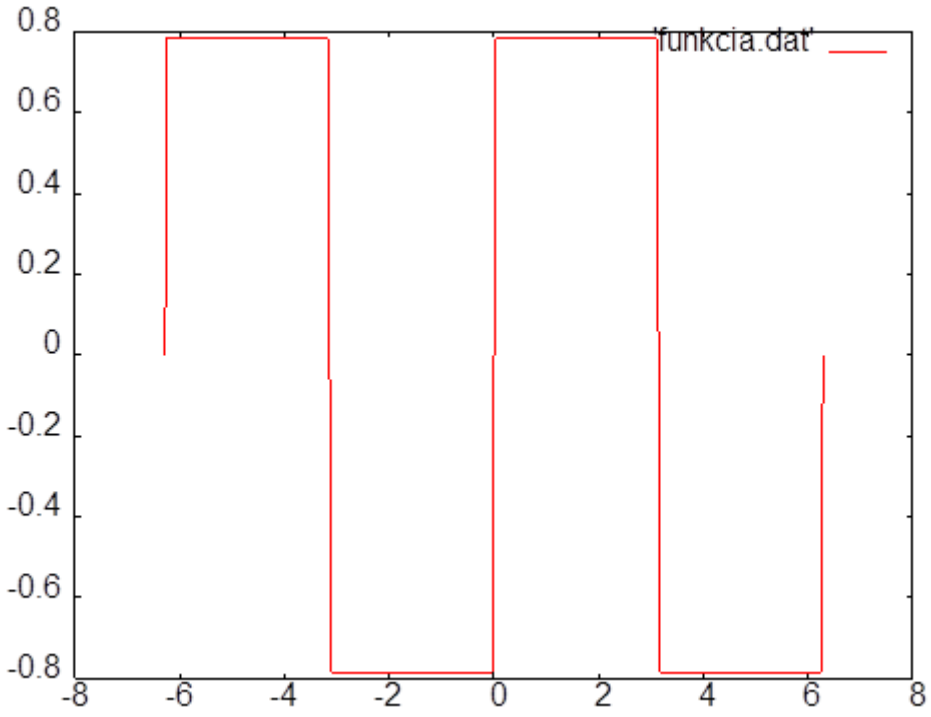
Skúsime sčítať 50 členov radu:



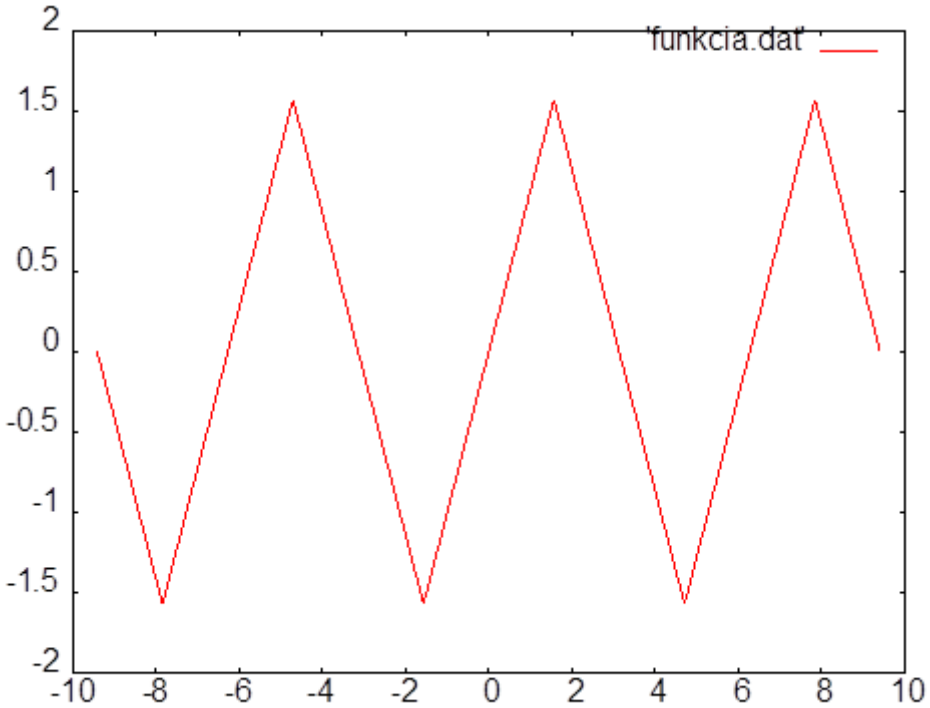
Ešte stále vidno v blízkosti hrán "prekmity". Preto zvýšime počet členov na 500:



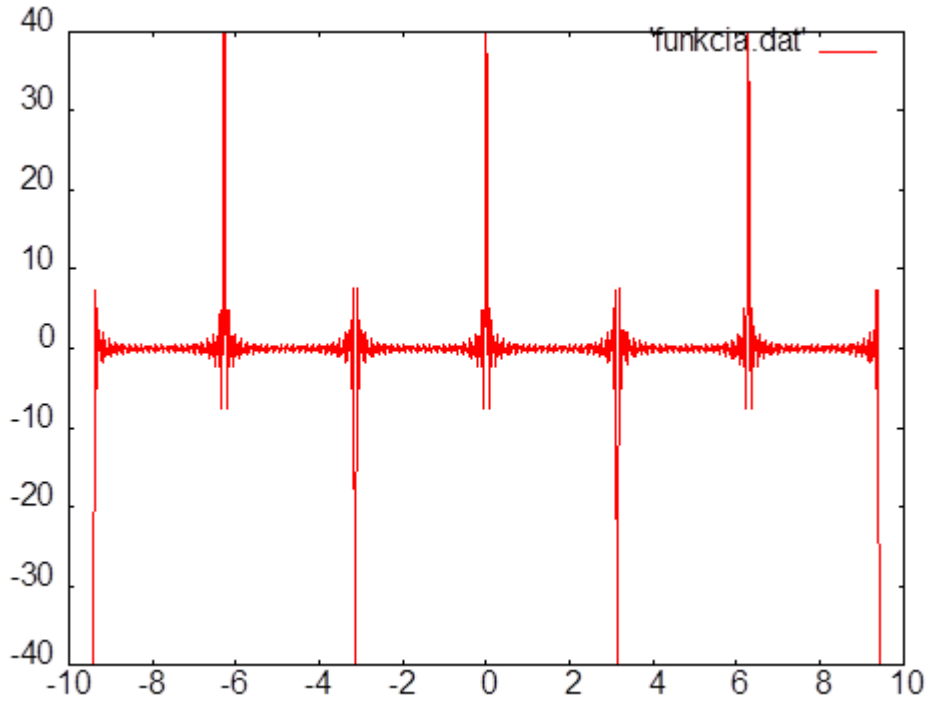
Už je to takmer dokonalý obdĺžnikový signál, má ale trochu zaoblené hrany. Skúsime preto 5000 členov:



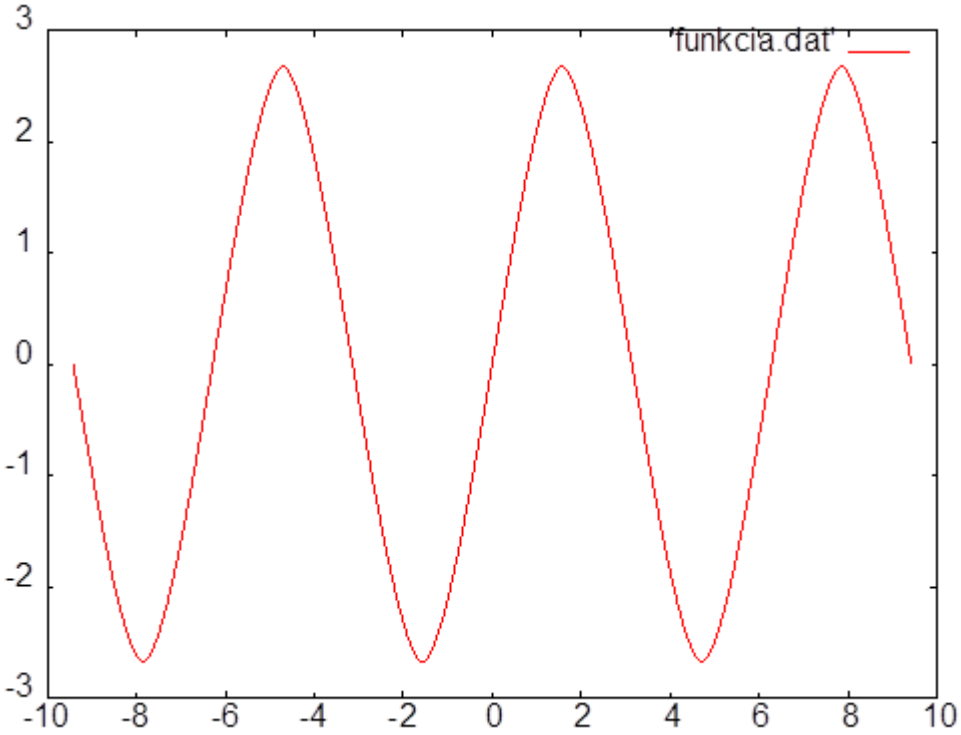
Vidíme, že ak má signál ostré hrany, Fourierov rad konverguje veľmi pomaly.



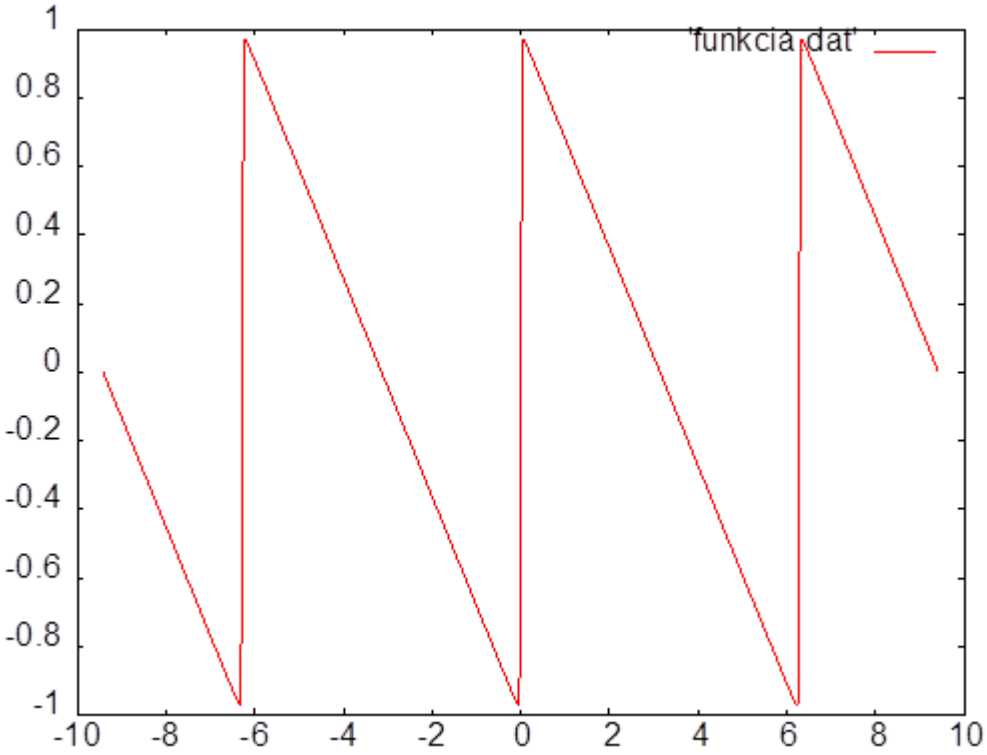
Graf funkcie $f_1(x)$



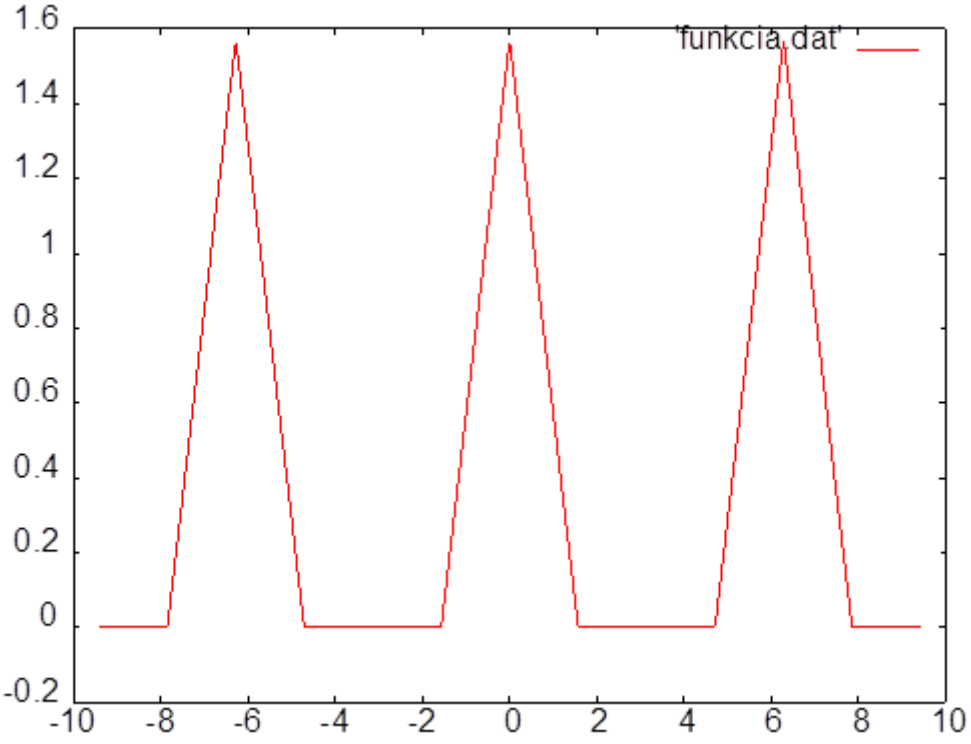
Graf funkcie $f_2(x)$



Graf funkcie $f_3(x)$



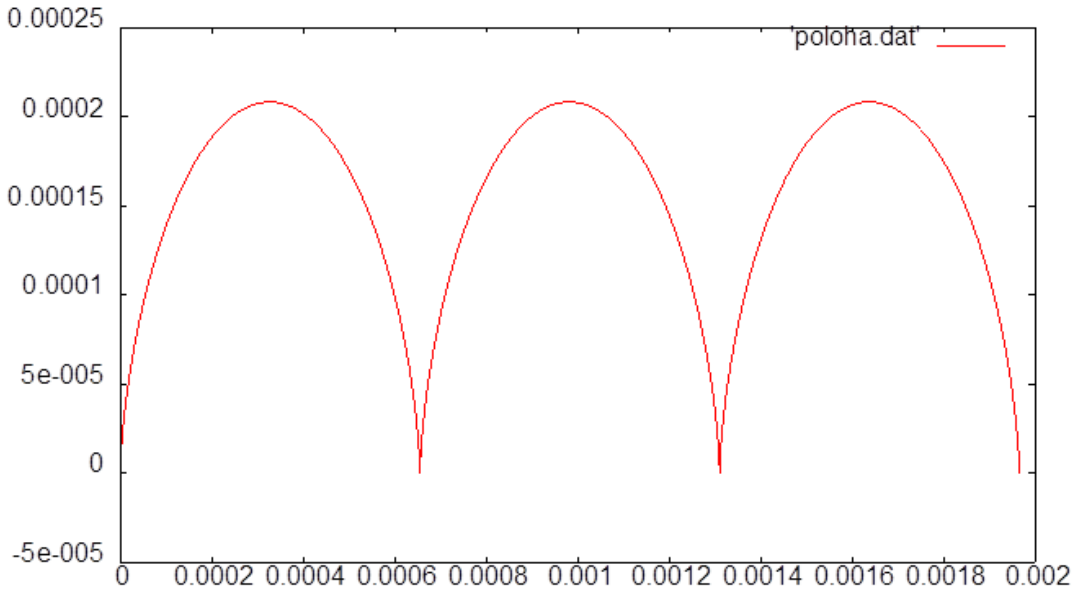
Graf funkcie $f_4(x)$



Graf funkcie $f_5(x)$

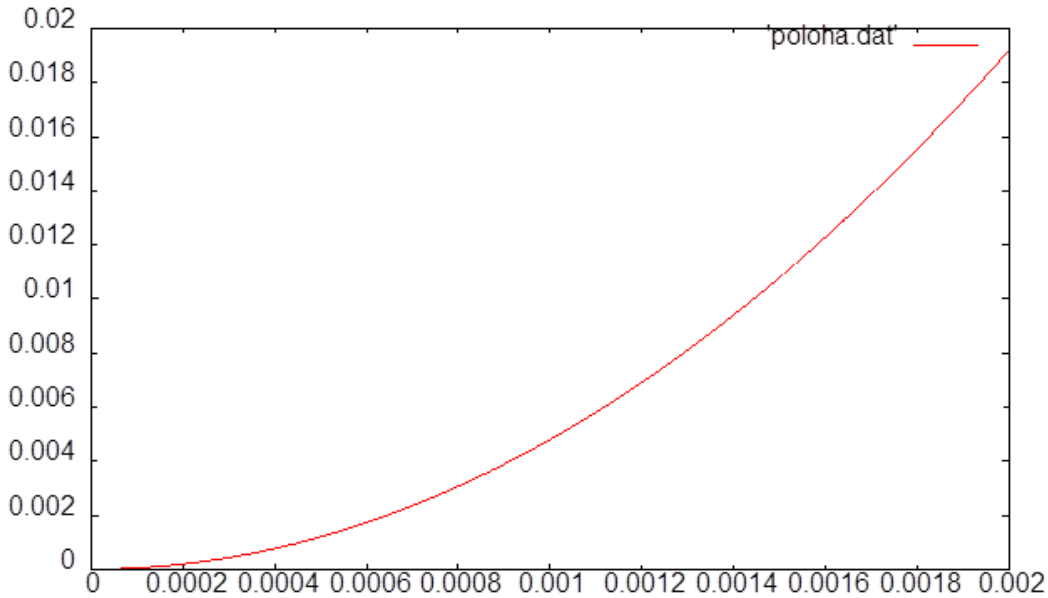
10.3. Pohybové rovnice

Dráha protónu pri počiatočných podmienkach uvedených v programe:



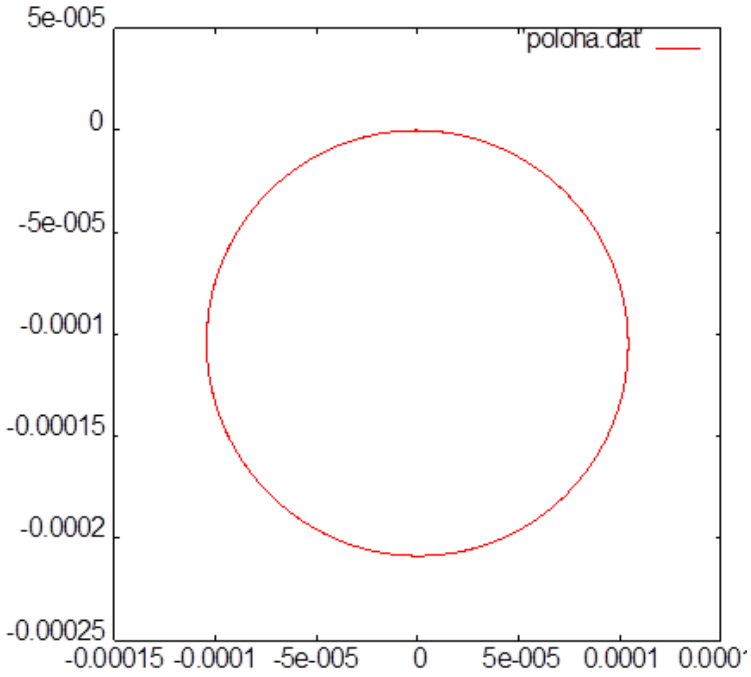
10.4. Pohyb protónu v elektrickom poli

Pohyb je po parabolickej dráhe:



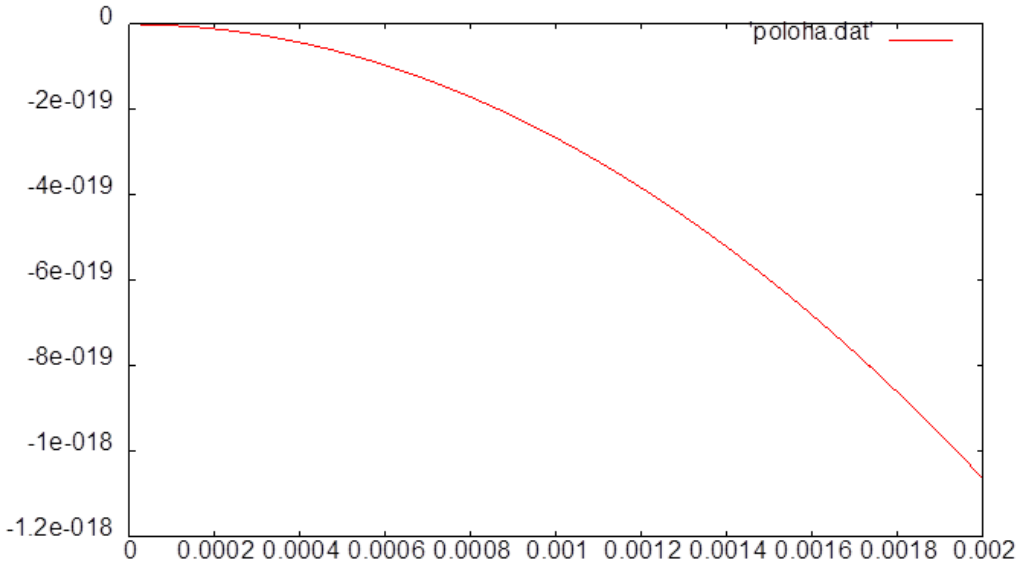
10.5. Pohyb protónu v magnetickom poli

Pohyb je po kruhovej dráhe:

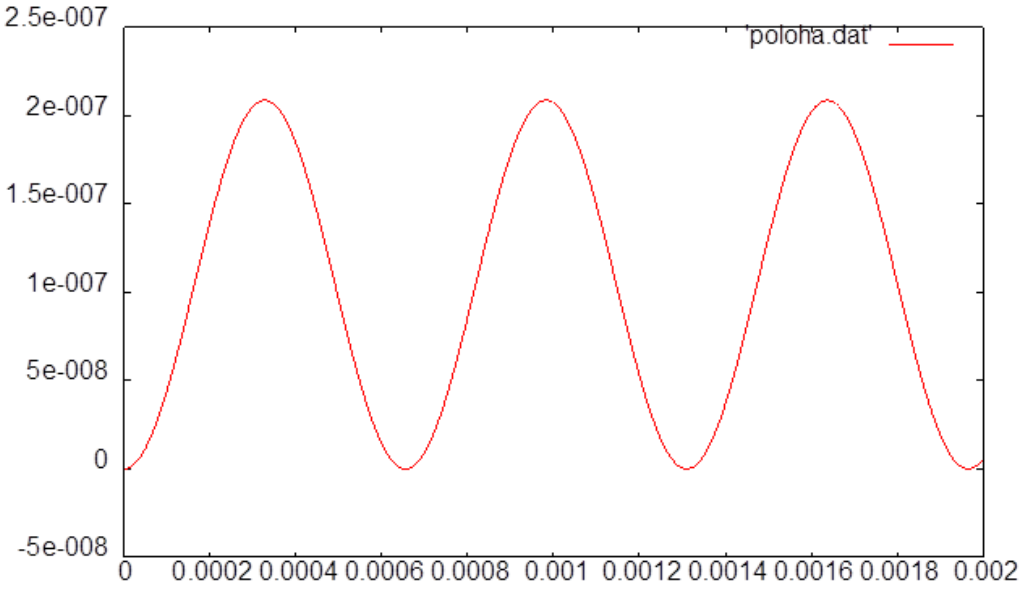


10.6. Pohyb protónu rýchlosťou $v_x = E/B$

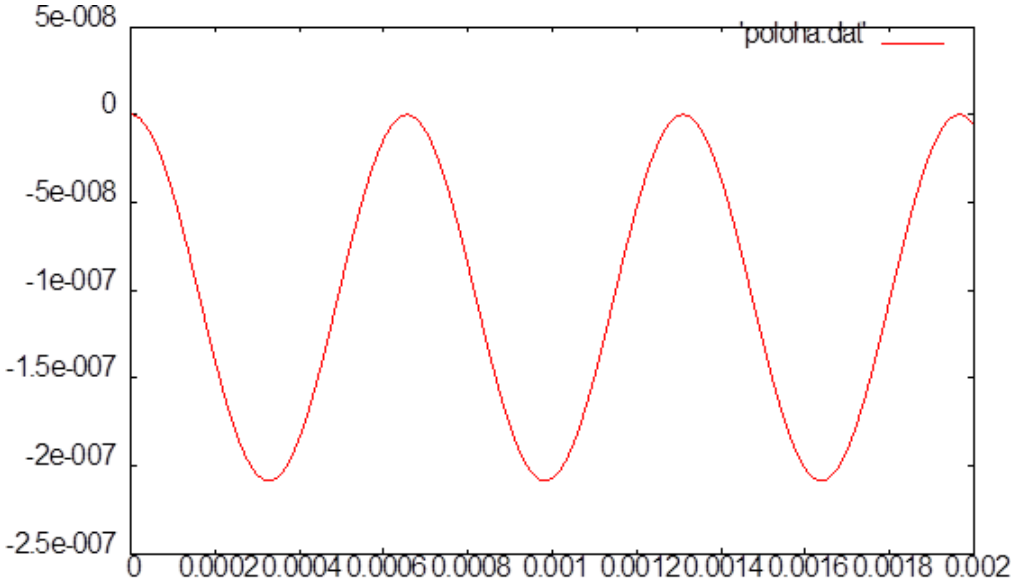
Vypočítaná dráha je v dôsledku zaokrúhľovania parabolická (mala by byť priamočiara), treba si však všimnúť aj jednotky na osiach (rozsah vertikálneho pohybu je prakticky nulový)::



Dráha pri 999 m/s, rýchlosť protónu je na začiatku menšia než optimálna, prevládne elektrická sila, dráha protónu je na začiatku zakrivená v kladnom smere osi *y*.:

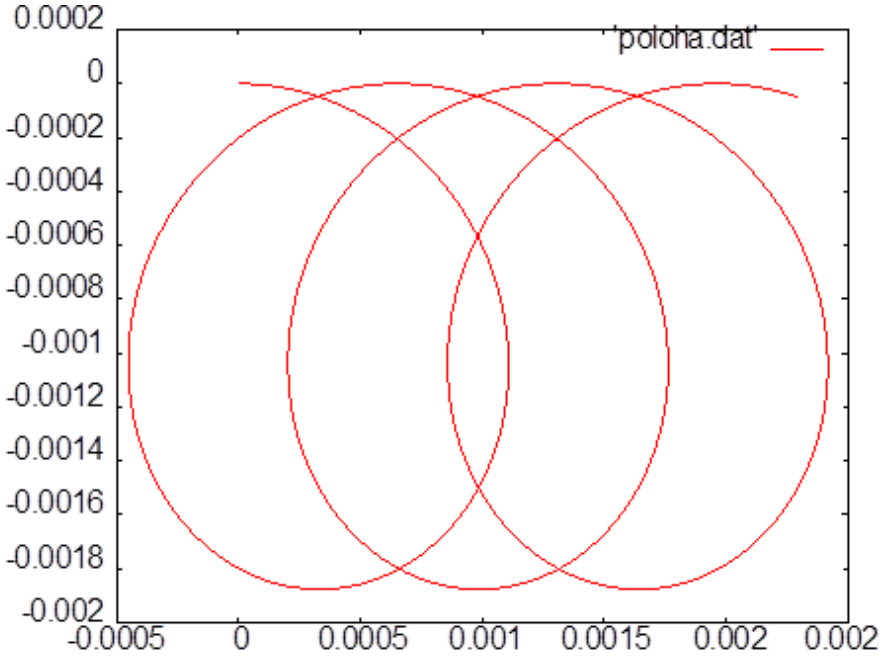


Dráha pri 1001 m/s, rýchlosť protónu je na začiatku väčšia než optimálna, prevládne magnetická sila, dráha protónu je na začiatku zakrivená v zápornom smere osi *y*.:

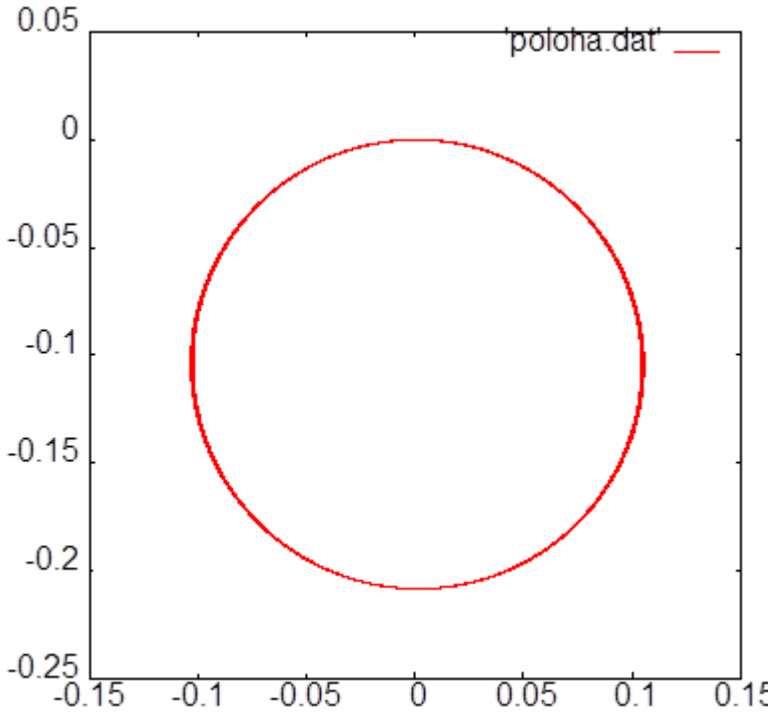


10.7. Pohyb protónu rýchlosťou $v_x = 10\,000\ m/s$

Dráha pri 10 000 m/s:

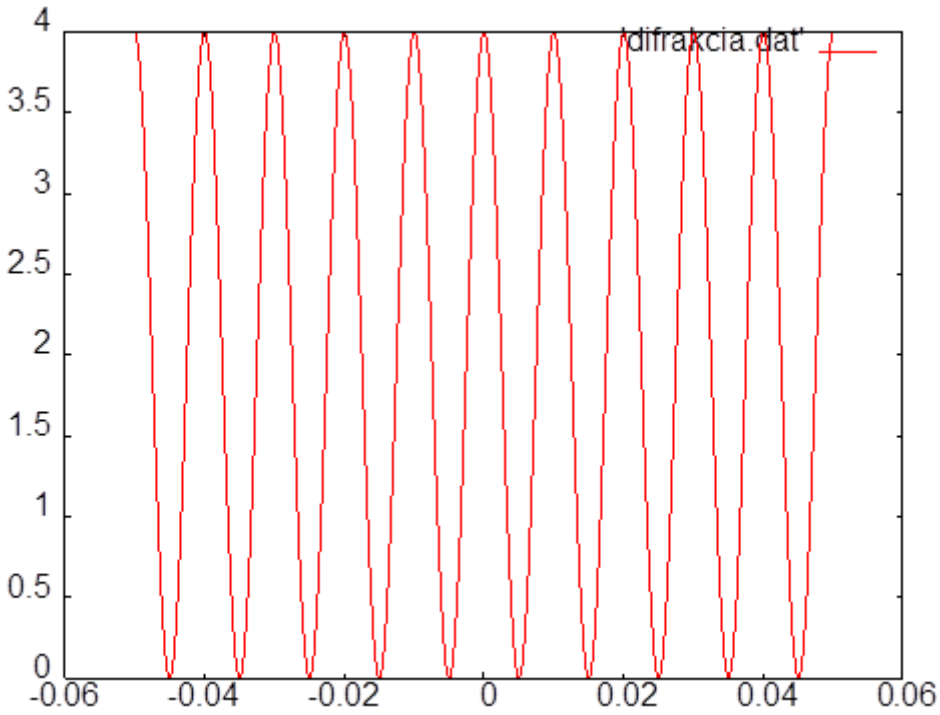


Pri 1 000 000 m/s bude prevládať magnetická sila (je úmerná rýchlosti), takže dráha sa zmení prakticky na kruhovú:

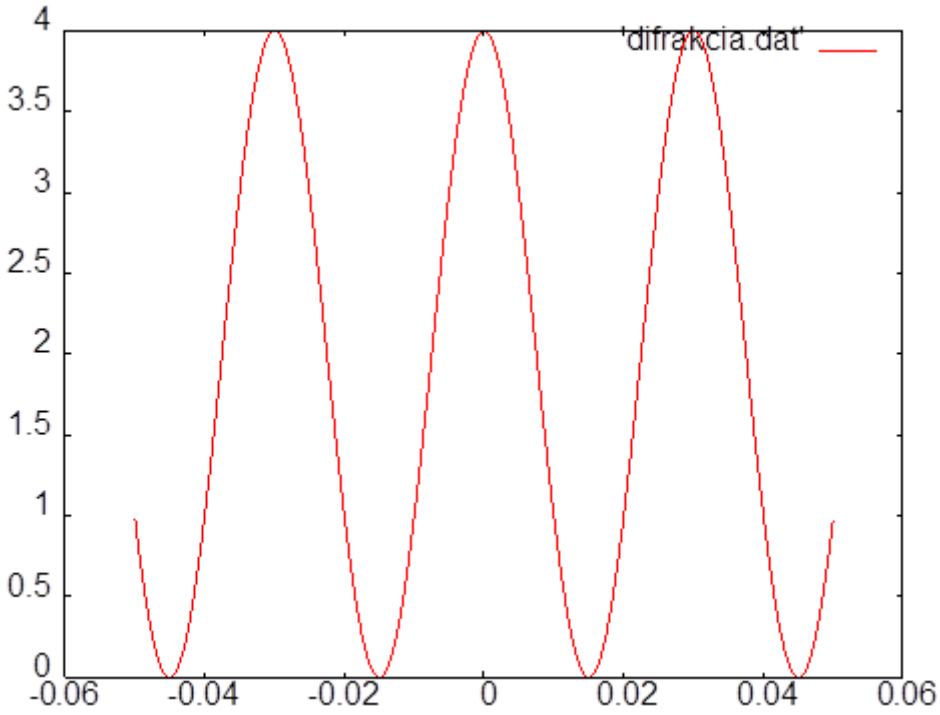


11.4. Interferencia vĺn na dvojštrbine

Pri približovaní štrbín sa prúžky zrieďujú:



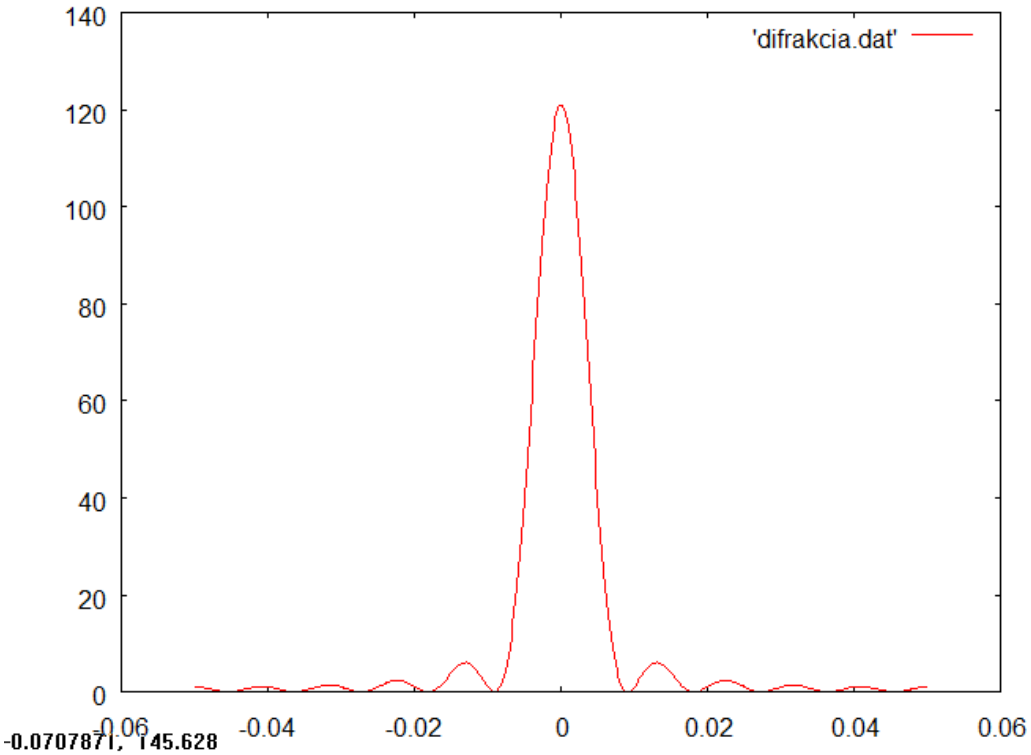
Interferenčné prúžky pri vzdialeností štrbín 60 μm



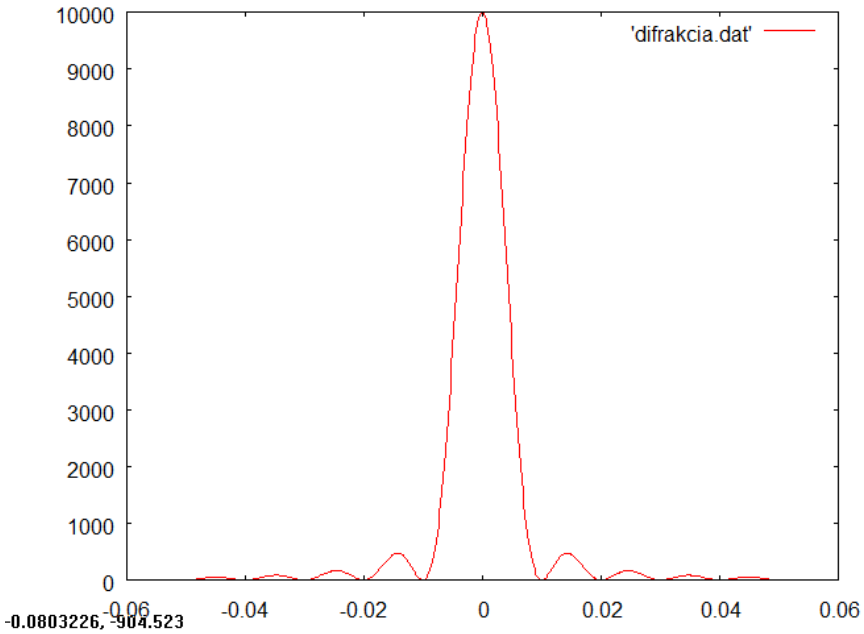
Interferenčné prúžky pri vzdialeností štrbín 20 μm

11.5. Interferencia vĺn na jednej (širšej) štrbine

Pri odporúčanom kroku 10 *lambda* vyzerá graf takto:

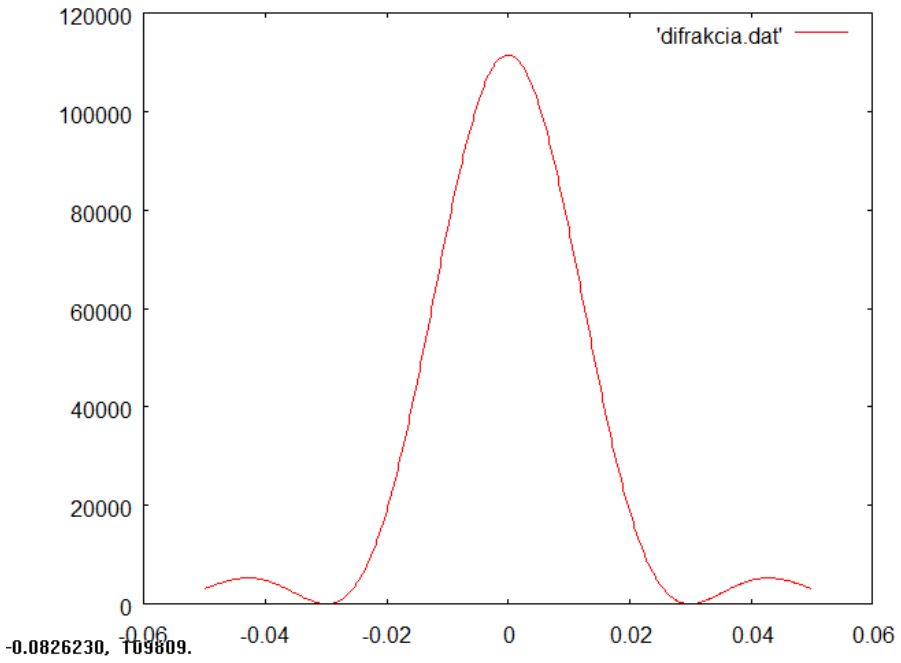


Graf vyzerá "hladko", ale polohy miním nie sú správne (nie sú násobkom 0,01 m). Zjemnením kroku na 1 *lambda* už dostávame správny výsledok:



Difrakcia na štrbine šírky 60 µm, krok výpočtu 1 lambda

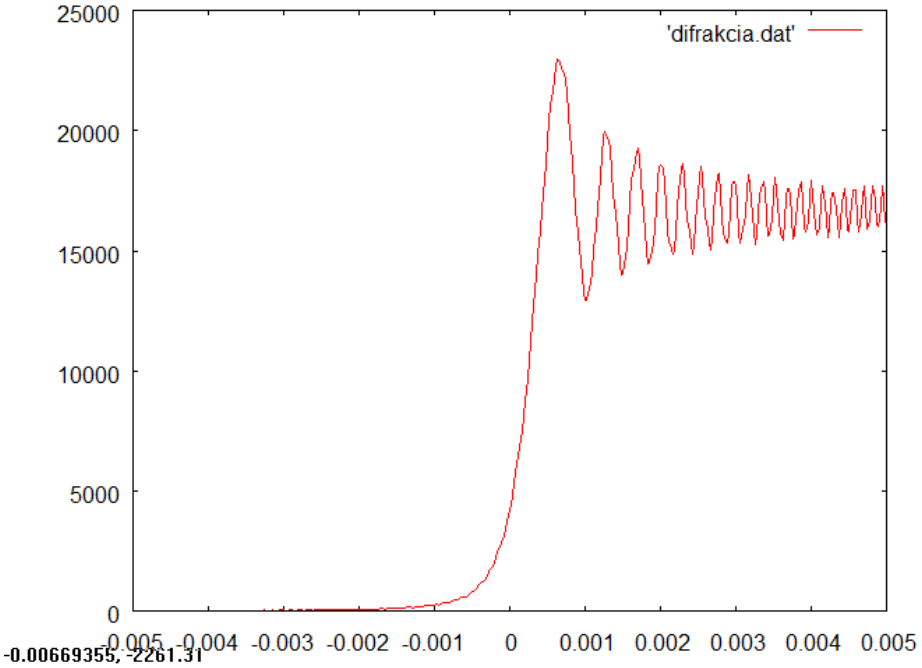
Zúžením štrbiny alebo zväčšením vlnovej dĺžky sa obrazec úmerne rozšíri:



Difrakcia na štrbine šírky 20 µm, krok výpočtu 0,1 lambda

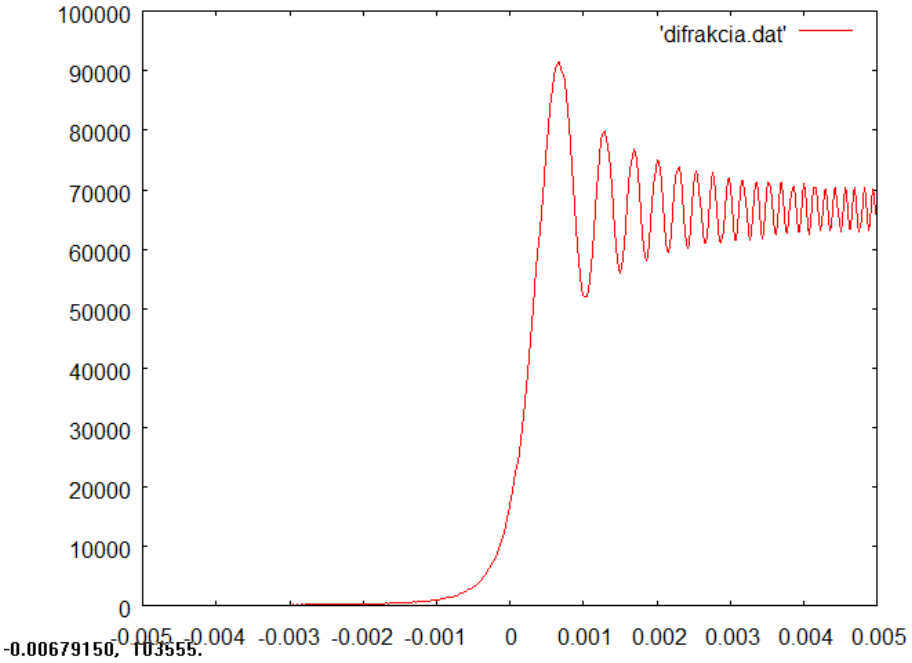
11.6. Ohyb svetla na hrane

Vypočítaný difrakčný obrazec je v pravej časti dosť nepravidelný:



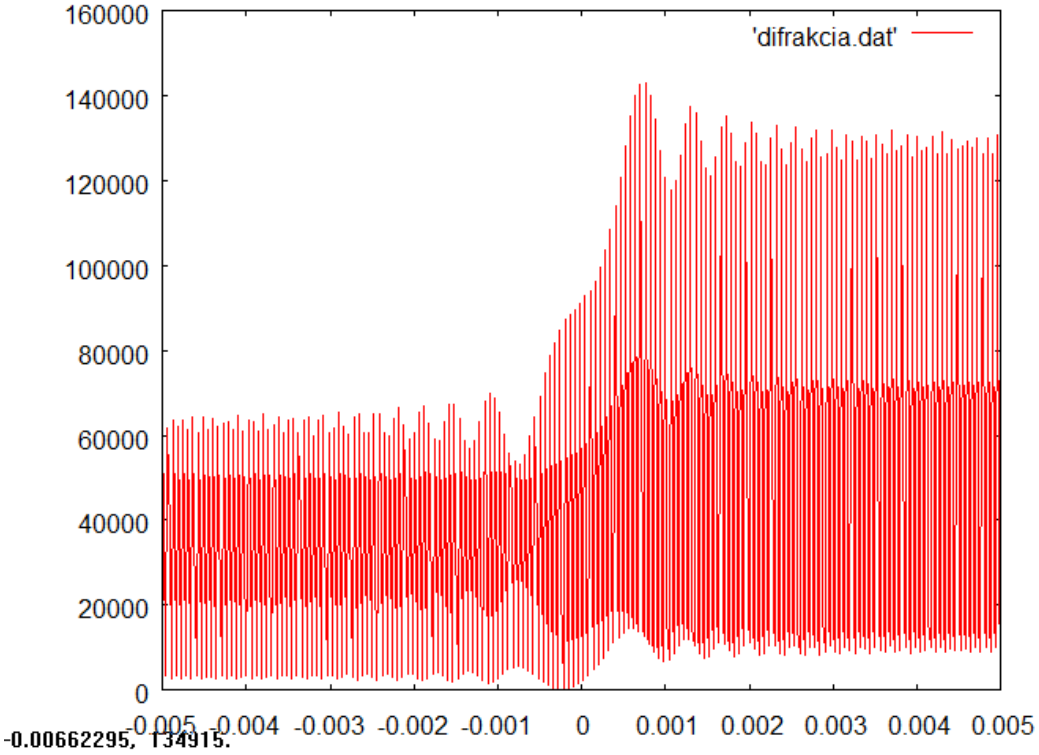
Difrakcia na hrane, krok výpočtu 10 lambda, rozsah 5 ymax

Po zjemnení kroku na 5 *lambda* a "nekonečna" na 20 *ymax*, je obrázok lepší:



Difrakcia na hrane, krok výpočtu 5 lambda, rozsah 20 ymax

Pozor! Veľkosť "nekonečna" a dĺžka kroku musia navzájom korešpondovať. Ak ponecháme dĺžku kroku 10 *lambda* a iba 10x zväčšíme "nekonečno" na 50 *lambda*, výsledok sa paradoxne zhorší:



Difrakcia na hrane, krok výpočtu 10 lambda, rozsah 50 ymax

Z nakresleného obrázku odčítame, že podiel intenzity svetla pre $y = 0$ a priemernej intenzity svetla na osvetlenej časti je asi 0,26. "Na hrane" je teda intenzita svetla iba štvrtinová, nie polovičná, ako by sa mohlo intuitívne očakávať.

12.2. Zobrazovacia rovnica tenkej spojnej šošovky

Kontrolný výpočet:

```
Zadaj a1: 10
b1 = 15
Z1 = -1.5

Process returned 0 (0x0)   execution time : 26.831 s
Press any key to continue.
```

12.3. Sústava dvoch tenkých spojných šošoviek

Výpočet pre lúče vychádzajúce z ohniska prvej šošovky:

```
Zadaj a1: 6
b2 = 4
Z = nan

Process returned 0 (0x0)   execution time : 3.241 s
Press any key to continue.
```

Kontrolný výpočet:

```
Zadaj a1: 9
b2 = 3.2
Z = -0.4

Process returned 0 (0x0)   execution time : 8.575 s
Press any key to continue.
```

12.4. Hlavné roviny optickej sústavy

Ľahko nájdeme (sledujeme hodnotu zväčšenia), že $Z = 1$ dosiahneme pri $a1 = -1{,}5$, kedy $b2 = -1$. Poloha hlavných rovín teda je: $h1 = -1{,}5$ cm a $h2 = -1$ cm. Obe hlavné roviny sa nachádzajú v priestore medzi šošovkami.

```
Zadaj a1: -1.5
b2 = -1
Z = 1

Process returned 0 (0x0)   execution time : 61.232 s
Press any key to continue.
```

12.5. Ohnisková vzdialenosť optickej sústavy

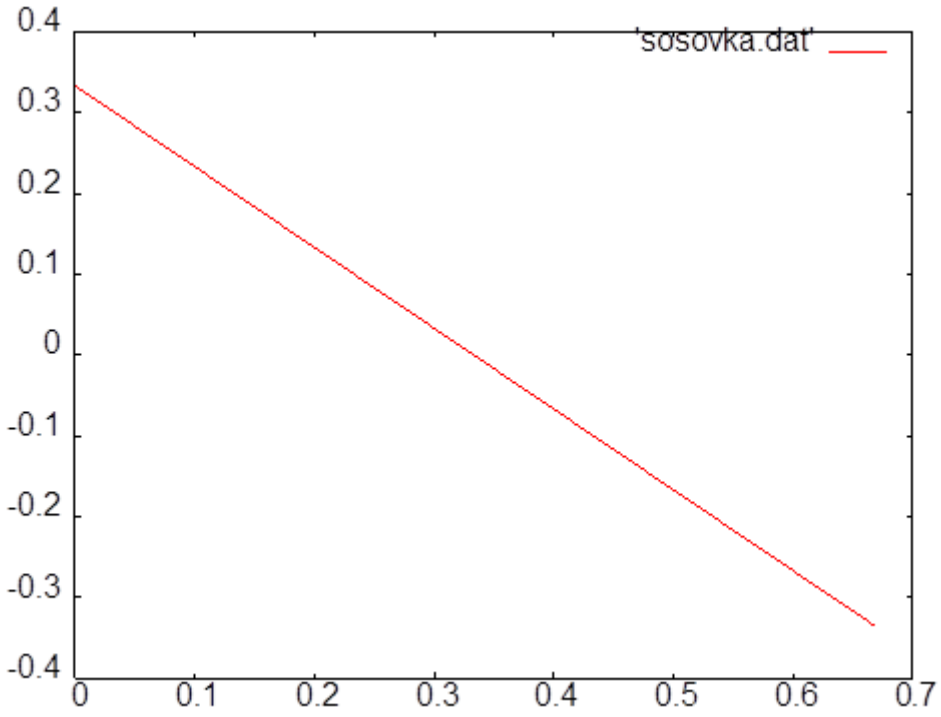
Zadaním veľkej hodnoty $a1$ (napríklad $a1$ = `1e100`) získame $b2 = 2$, čiže $o2 = 2$ cm. Skusmo nájdeme, že $b2$ je nekonečne veľké pri $a1 = 1{,}5$, čiže $o2 = 1{,}5$ cm.

```
Zadaj a1: 1e100
b2 = 2
Z = -3e-100

Process returned 0 (0x0)   execution time : 3.693 s
Press any key to continue.
```

```
Zadaj a1: 1.5
b2 = inf
Z = -inf

Process returned 0 (0x0)   execution time : 18.220 s
Press any key to continue.
```

12.6. Zobrazovacia rovnica optickej sústavy

Grafom je klesajúca priamka so smernicou -1:



Vidíme, že os y priamka pretína v bode 0,33, odkiaľ f = 1/0,33 = 3 cm. Hodnota je v zhode s výpočtom:

f = $o1$ - $h1$ = 1,5 cm – (–1,5 cm) = 3 cm, resp. f = $o2$ – $h2$ = 2 cm – (–1 cm) = 3 cm.

Sústava šošoviek sa teda správa rovnako, ako jedna šošovka s ohniskovou vzdialenosťou 3 cm.

13.2. Perióda kmitov kyvadla pre malé výchylky

Zistíme, že pri malých kmitoch je perióda kmitov $T1$ = 1,74995 s. Za 24 hodín urobí kyvadlo 86400/1,74995 = 49372,8 kmitov.

```
Perioda pri malych kmitoch = 1.74995 s

Process returned 0 (0x0)   execution time : 0.069 s
Press any key to continue.
```

13.3. Nastavovanie periódy kmitov

Nastavením x = 0,125 zistíme, že $T1$ je teraz 1,74892 s. Skrátené kyvadlo urobí za 24 hodín 86400/1,74892 = 49401,9 kmitov. Urobí teda o 29,1 kmitov viac. Hodiny teda nadbehnú o 29,1 . 1,75 s = 51 s.

```
Perioda pri malych kmitoch = 1.74892 s

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

13.5. Perióda kyvadla pri veľkých výchylkách

Pri amplitúde kmitov 0,1 rad je perióda 1,75106 s (dlhšia než teoretická).

```
Perioda pri malych kmitoch = 1.74995 s
0.87552 -0.0999899335
Skutocna perioda = 1.75106 s

Process returned 0 (0x0)   execution time : 98.978 s
Press any key to continue.
```

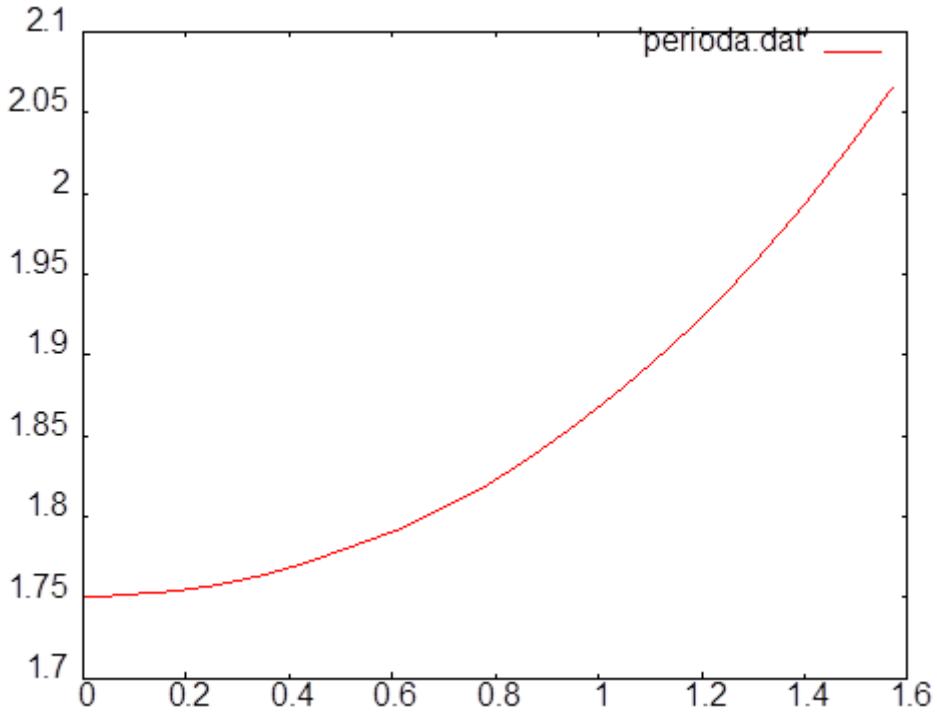
Pri amplitúde kmitov 90° je perióda 2,0655 s, čiže výrazne dlhšia.

```
Perioda pri malych kmitoch = 1.74995 s
1.03276 -1.5708929815
Skutocna perioda = 2.06554 s

Process returned 0 (0x0)   execution time : 159.064 s
Press any key to continue.
```

Odchýlku menej než 0,0001 s docielime pri amplitúde kmitov asi 1°, kedy je perióda 1,7500 s.

```
Perioda pri malych kmitoch = 1.74995 s
0.87499 -0.0174533125
Skutocna perioda = 1.75 s

Process returned 0 (0x0)   execution time : 129.008 s
Press any key to continue.
```

13.6. Závislosť periódy kyvadla od amplitúdy kmitov



V grafe je doplnená perióda kmitov pre nulovú amplitúdu pomocou teoretického vzťahu.

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <cstdlib>
#include <complex>

using namespace std;

int main()
{
    ofstream fout("difrakcia.dat");
    if(!fout)
    {
        cout << "Neda sa zapisovat do suboru difrakcia.dat" <<
endl;
        return 1;
    }

    double L=1;
    double d=60e-6;
    double lambda=600e-9;
    double k=2*M_PI/lambda;
    double ymin=-0.05;
    double ymax=0.05;
    double step=(ymax-ymin)/500;


    for(double y=ymin;y<=ymax;y+=step)
    {
        complex <double> amplituda(0,0);
        for(double yy=-d/2;yy<=d/2;yy+=10*lambda)
        {
            double r=sqrt(pow(L,2)+pow(y-yy,2));
            complex <double> faza(0,-k*r);
            amplituda+=exp(faza);
        }

        double intenzita = norm(amplituda);
        fout << y << "\t" << intenzita << endl;
        "\t" << intenzita << endl;



su ulozene v subore difrakcia.dat" << endl;
 -persist -e \"set style data lines; plot
        'difrakcia.dat'\"");
    return 0;
}
```