Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra aplykované matematiky

Bakalářská práce

Interaktivní webová demonstrace Fermiho akcelerace

Vladimír Kotýnek

Vedoucí práce: Ing. Tomáš Kalvoda, Ph.D.

17. května 2013

Poděkování

Chtěl bych tímto poděkovat panu Ing. Tomáši Kalvodovi, Ph.D. za navržení tohoto zajímavého tématu, poskytování cenných rad a celkově za výborné vedení mé práce. Dále bych chtěl poděkovat své rodině, která mě po celou dobu studia plně podporuje a svým přátelům, kteří jsou pro mě rovněž významnou duševní oporou.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 17. května 2013 .....

Odkaz na tuto práci

Kotýnek, Vladimír. *Interaktivní webová demonstrace Fermiho akcelerace.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2013.

Abstract

This thesis deals with creation of an interactive web demonstration of *Fermi acceleration*. At first an algorithm computing a time evolution of *Fermi acceleration* is designed and implemented. Then posibilities of creating interactive web demonstrations in computer algebra systems *Wolfram Mathematica* and *Sage* are reviewed and the interactive web demonstrations are created using both of them.

Keywords Fermi acceleration, Computer algebra system, Interactive web demonstration, Wolfram Mathematica, Wolfram Demonstrations, CDF, MathLink, Manipulate, Sage, Interact, Sage Cell Server, Cython, CTypes

Abstrakt

Tato práce se zabývá tvorbou interaktivní webové demonstrace *Fermiho akcelerace*. Nejdříve je navržen a implementován algoritmus vypočítávající časový vývoj *Fermiho akcelerace*. Poté jsou analyzovány možnosti tvorby interaktivních webových rozhraní pomocí počítačových algebraických systémů *Wolfram Mathematica* a *Sage* a pomocí obou těchto programů jsou interaktivní webové demonstrace vytvořeny.

Klíčová slova Fermiho akcelerace, Počítačový algebraický systém, Interaktivní webová demonstrace, Wolfram Mathematica, Wolfram Demonstrations, CDF, MathLink Sage, Interact, Sage Cell Server, Cython, CTypes

Obsah

Seznam obrázků

Úvod

Fermiho akcelerace

Fermiho akcelerace je název používaný pro originální mechanismus, jímž se italský fyzik Enrico Fermi [1] v roce 1949 pokoušel vysvětlit vznik vysokoenergetického kosmického záření. Později se zkoumání tohoto mechanismu věnoval, mimo jiných [3], i matematik Stanislaw Ulam [17], který v roce 1961 představil *Fermiho-Ulamův model* (FUM). Mechanismus FUM lze popsat jako pohyb bodové částice mezi dvěma stěnami nekonečných rozměrů, z nichž jedna je nehybná a druhá se periodicky pohybuje. Tím se mění vzdálenost mezi nimi, kterou musí částice urazit a při odrazu od pohyblivé stěny se mění i energie částice.

Zkoumání *Fermiho akcelerace* se obvykle zaměřuje na kvalitativní vlastnosti tohoto dynamického systému. Zaznamenávány jsou přitom změny rychlosti částice a časy, v nichž k těmto změnám došlo, vztažené na periodu pohybové funkce pohybující se stěny. V úvahu jsou přitom brány různé funkce s rozdílnou diferencovatelností. Pro lepší názornost získaných výsledků bývají hodnoty vynášeny do grafů znázorňujících vztah mezi rychlostí částice a pozicí pohyblivé stěny.

Často je zapotřebí vynášet do grafu stovky tisíc, či miliony bodů reprezentujících čas odrazu částice od stěny a její rychlost v onom okamžiku. V dobách vzniku FUM prováděly výpočet těchto dat obrovské halové počítače po dlouhé hodiny, ovšem s technologií dnešní doby můžeme tyto grafy generovat interaktivně za dobu nejvýše několika sekund například pomocí webové aplikace. Generování dat se tak stává mnohem komfortnější, než dříve.

V dnešní době existuje množství *počítačových algebraických systémů* (anglicky *Computer Algebra System* – CAS), ať už komerčních, či jako Open Source. Tyto zpravidla komplexní programy bývají účinným nástrojem k počítání náročných (ať už z důvodu komplikovanosti výpočtů nebo kvůli velkému množství početních operací) úloh z oblasti matematiky, fyziky, chemie, ekonomie a mnohých dalších odvětví vědy. Některé z těchto programů

začínají pronikat také do oblasti webových technologií a nabízejí různé způsoby, jak jejich výpočty prezentovat online. Nasnadě tedy je pokusit se využít potenciál CAS k vytvoření interaktivní webové demonstrace *Fermiho akcelerace*, čemuž ovšem bude muset předcházet analýza jejich možností v oblasti interaktivních ovládacích prvků a vytváření webových rozhraní.

Autorovou motivací k výběru tvorby interaktivní webové demonstrace *Fermiho akcelerace* jako tématu bakalářské práce byla zejména atypičnost tohoto zadání. Především nechtěl vytvářet běžný podnikový informační systém, nebo jinou standardní webovou aplikaci, ale vyzkoušet si něco nového, s čím se pravděpodobně v profesním životě nebude setkávat každý den.

Cíle práce

Prvním cílem této práce bylo analyzovat možnosti tvorby interaktivních webových demonstrací pomocí CAS *Wolfram Mathematica*, jakožto reprezentanta komerčních produktů na poli CAS. Program *Wolfram Mathematica* je rovněž hojně využíván ve výuce na FIT ČVUT v Praze.

Druhým cílem bylo provdení obdobné analýzy CAS *Sage*, jakožto reprezentanta Open Source, který bohužel není natolik rozšířen. *Sage* není v rámci výuky na FIT ČVUT v Praze běžně používán, zabývá se jím zde jen několik nadšenců.

Třetím cílem bylo navržení algoritmu, který bude vypočítávat data pro výzkum kvalitativních vlastností FUM. Algoritmus má být schopen vypočítat data pro různé počáteční hodnoty a různé funkce popisující periodický pohyb pohyblivé stěny.

Čtvrtým cílem bylo vytvoření progam implementující navržený algoritmus. Tento program má být možné posléze použít k výpočtu dat pro interaktivní webovou demonstraci.

Pátým cílem této práce byl pokus o vytvoření interaktivních webových demonstrací *Fermiho akcelerace*. K vytvoření demonstrací byly použity CAS *Wolfram Mathematica* a *Sage*, využita byla implementace navrženého algoritmu. Poté, co byly obě varianty vytvořeny, byly vzájemě porovnány, a získané poznatky byly zaznamenány.

Struktura práce

První část této práce se věnuje návrhu algoritmu pro výpočet časového vývoje FUM a jeho implementaci (dále jen implementace) v jazyce C/C++. Nejprve je problém analyzován a jsou diskutovány možnosti řešení. Poté

následuje návrh algoritmů a funkcí, jenž jsou použity pro výpočet časového vývoje FUM. Nakonec je popsána implementace programu v jazyce C.

Druhá část této práce se věnuje analýze možností tvorby interaktivních webových demonstrací v CAS *Wolfram Mathematica* a rovněž analýze možnosti využití tohoto CAS k implementaci demonstrace. Představen je zde formát CDF a s ním i *Wolfram CDF Player*, dále *Wolfram Demonstrations Project* a tvorba interaktivních rozhraní pomocí příkazu `Manipulate`, v neposlední řadě je představen protokol *MathLink* a je poreferováno o možnostech použití kompilovaného kódu ve *Wolfram Mathematica*. Nakonec jsou krátce zmíněny možnosti, jak graficky prezentovat implementací vypočítané hodnoty.

Třetí část této práce se věnuje analýze možností tvorby interaktivních webových demonstrací v CAS *Sage* a také analýze možnosti využití tohoto CAS k implementaci demonstrace. Představen je zde *Sage* a jeho stručná historie, jelikož se autor práce domnívá, že na rozdíl od *Wolfram Mathematica* není tolik známý. Dále je zkoumán příkaz `interact`, s ním spojená komunita *Sage Interact* a *Sage Cell Server*, kde se práce věnuje také použití *Sage* na webu. V neposlední řadě je představen jazyk *Cython* a jako alternativní možnost pro použití jazyka C je uvedena knihovna *CTypes* pro jazyk Python. Nakonec jsou zmíněny možnosti, jak graficky prezentovat implementací vypočítané hodnoty.

Čtvrtá část této práce se věnuje vytváření interaktivní webové demonstrace v CAS *Wolfram Mathematica* a *Sage*. Nejdříve popisuje postup a užité funkce při tvorbě interaktivní demonstrace pomocí *Wolfram Mathematica*. Dále popisuje postup a užité funkce při tvorbě druhé verze interaktivní demonstrace v *Sage*. Nakonec tato část práce obě verze navzájem porovnává, hodnotí jejich výhody a nevýhody a představuje příklady jejich výstupů.

Kapitola 1

Návrh a implementace algoritmu

1.1 Popis problému

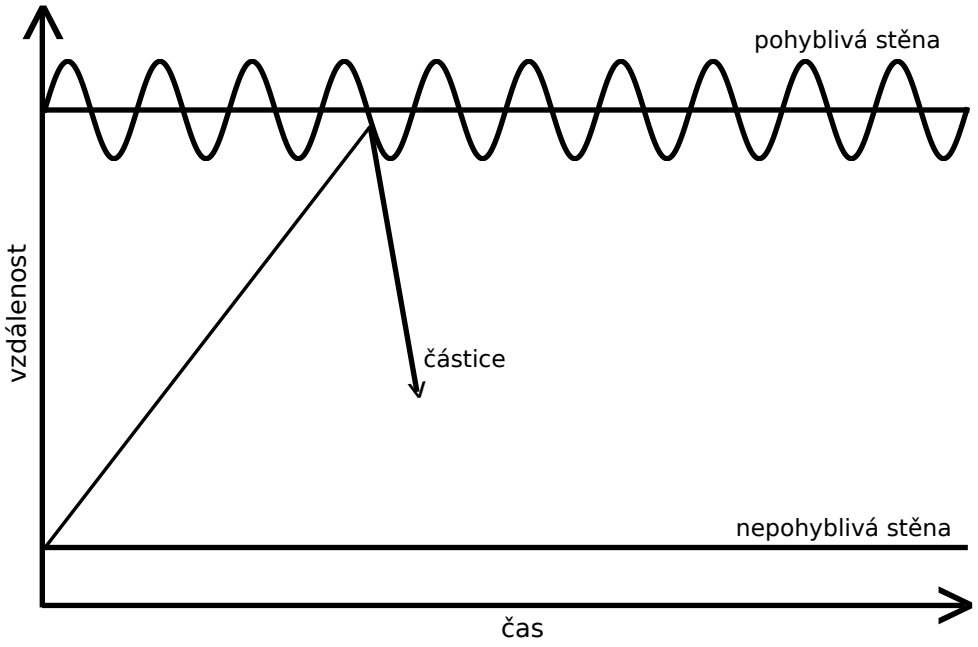
Variantou *Fermiho akcelerace* uvažovanou v této práci je zjednodušená verze *Fermiho-Ulamova modelu* (SFUM) [2]. Tento dynamický systém se skládá z bodové částice, která se pohybuje mezi dvěma nekonečnými stěnami představujícími magnetická zrcadla, od nichž se odráží, přičemž jedna z těchto dvou stěn je nepohyblivá, zatímco druhá se periodicky pohybuje, viz obrázek 1.1.

Pro výpočet rychlosti částice po $n + 1$ kolizích s pohybující se stěnou (dále jen kolize) platí vztah

$$v_{n+1} = |v_n + F'(t_{kolize})|, \tag{1.1}$$

kde v_{n+1}, rychlost částice po $n + 1$ kolizích, odpovídá absolutní hodnotě součtu v_n, rychlosti částice po n kolizích, a první derivace pohybové funkce pohybující se stěny ($F(t_{kolize})$) v odpovídající fázi pohybu t_{kolize}. Fáze času t_{kolize} odpovídá okamžiku kolize, tedy průsečíku funkce pohybu částice (lineární funkce času t) s funkcí pohybu stěny (periodická funkce času t), vzhledem k periodě pohybu stěny, při jejím výpočtu je nutno uvažovat dobu, po kterou poletí částice od pohybující se stěny k nepohyblivé, než se od ní odrazí, a poté dobu, po kterou poletí částice k pohybující se stěně po odrazu od stěny nepohyblivé. Je vhodné podotknout, že pohybová funkce nepohyblivé stěny je konstantní ($G(t) = 0$), tudíž její derivace v čase kolize stěny a částice je rovna nule a tudíž se velikost rychlosti částice kolizí s nepohyblivou stěnou nezmění, pouze její směr bude opačný, jedná se tedy o tzv. dokonale pružný odraz.

Úkolem tedy je navrhnout a implementovat funkci, která na základě zadaných vstupních parametrů vypočítá časový vývoj rychlosti částice. Vstup-



Obrázek 1.1: Model SFUM – závislost pozice částice na čase

ními parametry budou rychlost částice po poslední kolizi, odpovídající fáze pohybu pohybující se stěny v okamžiku poslední kolize částice s nepohyblivou stěnou, počet kolizí částice s nepohyblivou stěnou, pro které bude funkce výsledky počítat, granularita dat na výstupu a funkce pohybu stěny. Pro účely této práce bude postačovat, pokud tato funkce dokáže vypočítat data pro SFUM s pohyblivou stěnou pohybující se podle funkce sinus a pilové funkce, přičemž bude možno měnit minimální vzdálenost obou stěn od sebe a absolutní hodnotu amplitudy pohybové funkce pohyblivě stěny. Výstupem funkce pak bude pole dvourozměrných vektorů, jejichž jedna položka bude rychlost částice v okamžiku kolize s nepohyblivou stěnou a druhá bude perioda pohybu pohyblivé stěny v tomto okamžiku.

1.2 Návrh programu

1.2.1 Diskuze možností

Po nastudování problému bylo zjištěno, že funkce, jenž má být navržena, by měla vykonávat v zásadě dvě hlavní činnosti v cyklu po zadaný počet opakování. Zaprvé musí nalézt průsečík dvou známých funkcí a zadruhé dosadit tento průsečík do první derivace konkrétní pohybové funkce a pomocí výsledku vypočítat novou aktuální rychlost částice a čas příští srážky s nepohyblivou stěnou, resp. získat novou funkci pohybu částice.

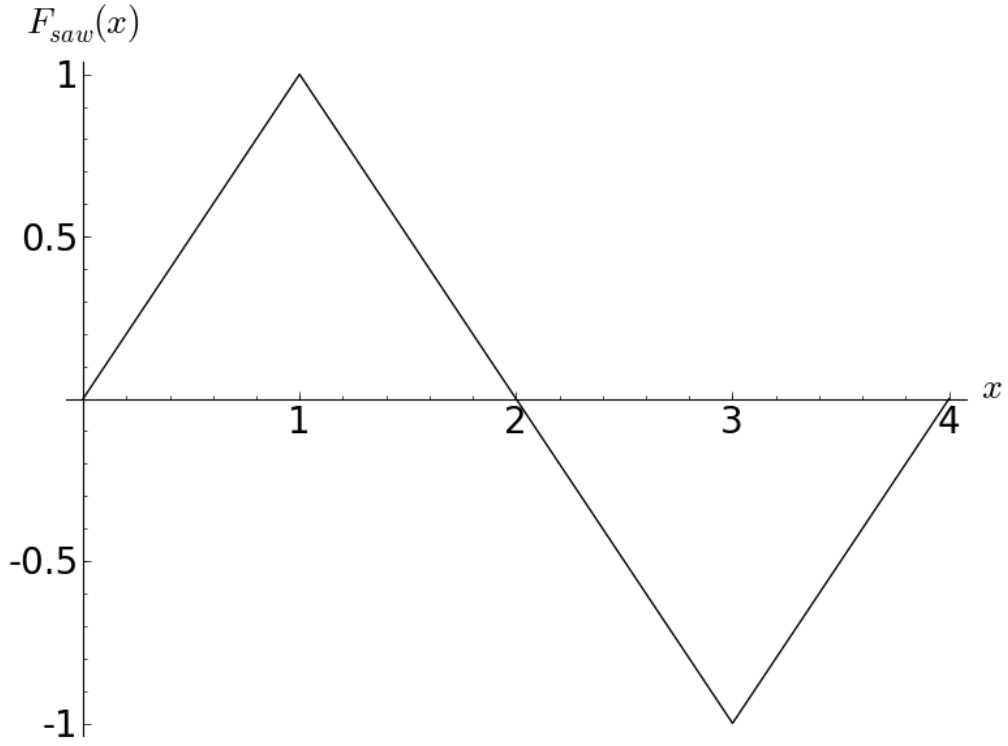
Derivaci funkce sinus, resp. $a + b \sin x$, kde parametry a a b jsou známy, spočítáme snadno jako $b \cos x$ a derivaci pilové funkce $a + b \cdot F_{saw}(x)$ lze také snadno definovat jako -1 pro funkci klesající (přibližuje se nepohyblivé stěně), -1 pro funkci rostoucí (vzdaluje se od nepohyblivé stěny) a 0 pokud je odpovídající pozice pohyblivé stěny právě ve vrcholu pily, kde sice tato funkce nemá derivaci, ovšem pro potřeby návrhu je tato derivace dodefinována jako $F'_{saw}(x) = 0$, což má i své fyzikální opodstatnění, neboť dojde-li v tomto okamžiku ke kolizi částice se stěnou, jeví se stěna jako nepohybující se.

Jako „pilová“ funkce je uvažována taková periodická funkce, jenž je na intervalech $\langle 0; 1\rangle$ a $\langle 3; 4\rangle$ rostoucí a s osou x svírá úhel $45\,^\circ$. Na intervalu $\langle 1; 3\rangle$ je pak klesající a s osou x svírá úhel rovněž $45\,^\circ$, viz obrázek 1.2.

Alternativní možností, jak získat hodnotu derivace funkce v daném bodě by mohlo být použití prostředků CAS, v němž bude tento implementace algoritmu využívána, což by ovšem ovlivnilo přenositelnost implementace algoritmu i na jiné CAS a navíc by rychlost výpočtu byla ovlivněna i latencí komunikačního spojení mezi implementací algoritmu a tímto CAS.

Pro nalezení průsečíku dvou funkcí byl nejdříve navržen a implementován jednoduchý krokovací algoritmus, který vzájemně porovnával hodnotu obou funkcí v určitém bodě, dokud nenarazil na shodu (viz obrázek 1.3). Při testování se ukázalo, že tento algoritmus funguje dostatečně rychle, tudíž bylo do jeho vylepšování ustoupeno.

Alternativou k navrženému algoritmu by mohla být například *Newtonova metoda (Newton-Raphsonova metoda)* pro řešení soustavy nelineárních rovnic $f(x) = 0$, jejíž implementace již existují [4]. Další alternativou by pak mohlo být použití funkcí CAS, v němž bude implementace tohoto algoritmu použita, ovšem zde stejně jako v případě hledání hodnoty první derivace funkce v daném bodě hrozí ovlivnění přenositelnosti implementace algoritmu a ovlivnění rychlosti výpočtu latencí při přenosu dat.



Obrázek 1.2: Ukázka průběhu „pilové“ funkce

1.2.2 Návrh algoritmu

Algoritmus pro nalezení doby srážky částice s pohybující se stěnou, popsaný na diagramu 1.3, postupuje následujícím způsobem:

1. Vypočítá pozici pohybové funkce pohyblivé stěny SFUM t_{start}, od níž se částice pohybuje v oblasti, kde může dojít ke srážce s pohyblivou stěnou. Tato fáze je dána přičtením doby, po kterou trvá částici překonání minimální vzdálenosti d_{min} mezi oběma stěnami, k fázi periody v okamžiku poslední kolize částice s nepohyblivou stěnou.

2. Inicializuje délku časového kroku pro testování t_{step} a nastaví se čas testování na $t_{test} = t_{start}$.

3. Vypočítá vzdálenost částice d_p od nepohyblivé stěny a vzdálenost pohyblivé stěny d_{wall} od nepohyblivé stěny a tyto vzdálenosti porovná.

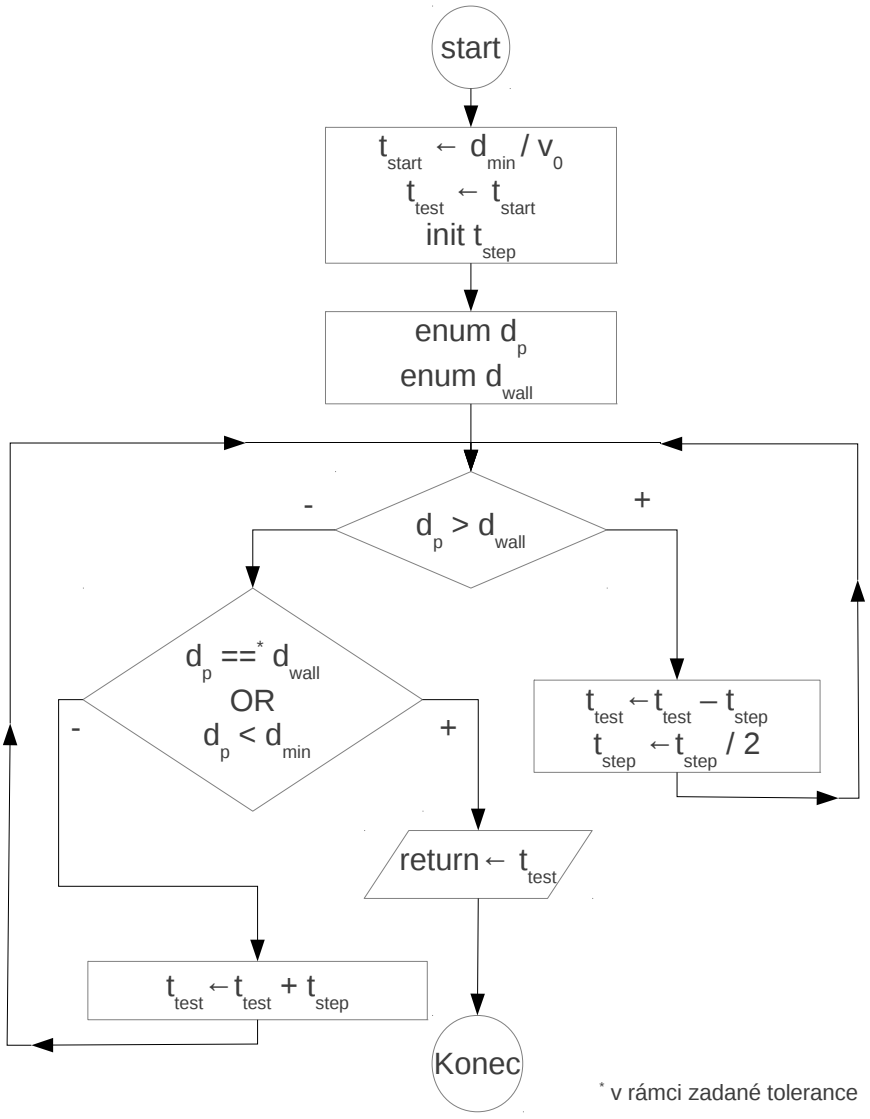
    - Pokud $d_p > d_{wall}$ (při prvním průchodu nemůže nastat), nastaví se čas testování na $t_{test} = t_{test} - t_{step}$ a dále se zmenší velikost

časového kroku na polovinu ($t_{step} = \frac{t_{step}}{2}$) a pokračuje se opět krokem 3.

- Pokud $d_p < d_{wall}$, nastaví se testování na $t_{test} = t_{test} + t_{step}$ (testování se posune o krok dále) a pokračuje se opět krokem 3.
- Pokud $d_p = d_{wall}$ v rámci zadané tolerance, znamená to, že byl nalezen průsečík obou funkcí v čase t_{test}.

Abychom zabránili zacyklení programu z důvodu omezené přesnosti strojových výpočtů, je nutné zavést pevnou hladinu tolerance při porovnáváni. V případě, že bychom tak neučinili, může nastat situace, že velikost časového kroku t_{step} bude tak malá, že již další půlení z hlediska reprezentace její hodnoty, jakožto čísla s plovoucí desetinnou čárkou, nezmění jeho hodnotu.

Složitost algoritmu ovlivňuje tolerance porovnávání, která ovlivní počet půlení velikosti časového kroku, a počet výpočtů a vzájemného porovnání pozic pohyblivé stěny a částice, které musí program vykonat, než nalezne přibližnou shodu. Při vhodné volbě velikosti časového kroku t_{step} při její inicializaci můžeme dosáhnout nejvýše desítek opakování výpočtu vzájemné pozice stěny a částice, než dojde k půlení velikosti časového kroku. Po každém zmenšení velikosti časového kroku provede algoritmus už vždy pouze jedno nebo dvě porovnání vzájemné pozice. Jedno v případě, že po přičtení nového časového kroku k aktuálně testované fázi periody bude platit $d_p \geq d_{wall}$, tudíž nastane rovnost, nebo bude opět opakováno zmenšení velikosti kroku. Dvě v případě, že po přičtení nového časového kroku bude platit $d_p < d_{wall}$, tudíž se krok přičítá podruhé, čímž se dostaneme na testovanou fázi periody, jíž jsme testovali před zmenšením časového kroku a dojde k jeho dalšímu zmenšování. Z tohoto plyne, že za konečný počet k provnávání vzájemné pozice částice a pohyblivé stěny dojde k nalezení okamžiku kolize částice s pohyblivou stěnou. Za elementární operaci je zde považován výpočet aktuálních poloh pohyblivé stěny v zadané fázi pohybu a částice v témže okamžiku a jejich vzájemné porovnání. Lze tedy prohlásit že asymptotická složitost tohoto algoritmu je nejhůře $O(k)$, kde pro k platí vztah (1.2), přičemž parametr t_p představuje periodu pohybu stěny, parametr t_{step_0} reprezentuje inicializační hodnotu časového kroku a *tolerance* je míra tolerance při porovnávání vzájemné pozice částice a pohyblivé stěny. V případě, že velikost t_{step_0}, t_p a hladiny tolerance jsou pevně určeny, lze prohlásit, že rovněž asymptotická složitost algoritmu pro nalezení průsečíku funkce pohybu částice s funkcí pohybu stěny je konstantní ($O(1)$). Ovšem v případě, že by byl algoritmus používán s větším množstvím periodických



Obrázek 1.3: Algoritmus pro nalezení první kolize částice s pohyblivou stěnou po kolizi se stěnou nepohyblivou.

funkcí s různou délkou periody, byla by složitost tohoto algoritmu $O(t_p)$.

$$k = 1 + \frac{t_p}{t_{step_0}} + \log_2\left(\frac{t_{step_0}}{\text{tolerance}}\right). \tag{1.2}$$

Algoritmus pro výpočet dat časového vývoje SFUM se skládá pouze z cyklického výpočtu okamžiků kolizí částice s pohyblivou a pevnou stěnou a ukládáním dat do výstupní proměnné předávané návratovou funkcí. Je však nutné si uvědomit, že po kolizi částice s pohyblivou stěnou je možné, že než tato částice opět koliduje se stěnou nepohyblivou, muže dojít ještě k dalším kolizím se stěnou pohyblivou a to v takových fázích pohybu stěny, kdy je vzdálenost částice od nepohyblivé stěny stále ještě větší nebo rovna minimální vzájemné vzdálenosti obou stěn.

Pro nalezení požadovaného počtu kolizí vykoná algoritmus následující kroky:

1. Nastaví fázi pohybu funkce při poslední kolizi s nepohyblivou stěnou t_0 na hodnotu ze vstupního parametu t a nastaví rychlost v okamžiku poslední kolize částice s nepohyblivou stěnou v_0 na hodnotu ze vstupního parametru v.

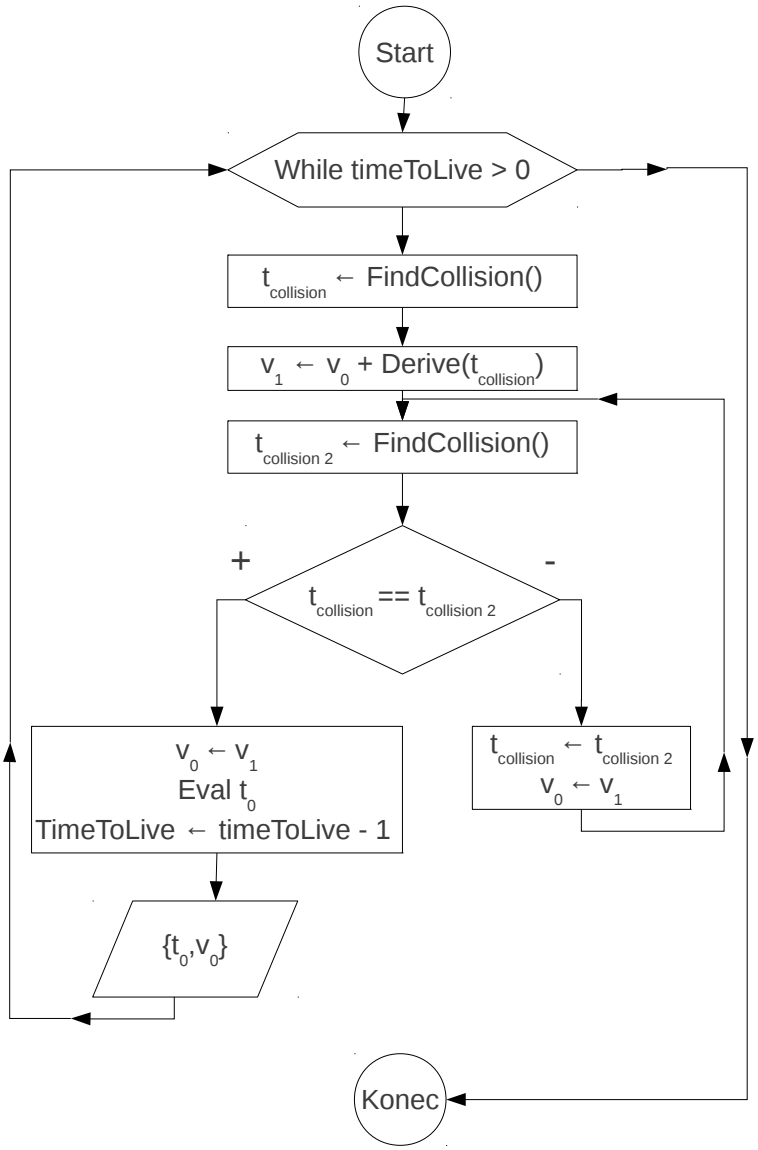
2. v cyklu pro n opakování, nebo dokud rychlost částice není rovna 0, kde n je počet kolizí částice s nepohyblivou stěnou, pro něž má funkce vypočítat data, budou vykonávány následující kroky:

   2.1. Algoritmem pro nalezení kolize částice s pohyblivou stěnou je vypočítána nejbližší následující kolize od fáze pohybové funkce t_0 při rychlosti v_0. Výslednou fáze pohybové funkce, kdy dojde ke kolizi, je označena t_1.

   2.2. Vypočítá se nová rychlost v_1 částice ve fázi pohybové funkce t_1.

   2.3. Od fáze pohybové funkce, kdy došlo k poslední kolizi částice s pohyblivou stěnou se upraveným algoritmem pro nalezení kolize částice s pohyblivou stěnou kontroluje, zda-li nedojede ke kolizi, než je částice tak blízko k nepohyblivé stěně, že již ke kolizi s pohyblivou stěnou dojít nemůže. Dojde-li zde ke kolizi částice s pohyblivou stěnou, je vypočítána nová rychlost v_1 částice a jako fáze pohybové funkce v okamžiku poslední kolize částice s pohyblivou stěnou t_1 je nastavena fáze pohybové funkce, při níž nyní došlo ke kolizi, a celý tento postup se opakuje.

   2.4. Vypočítá se fáze pohybové funkce, při níž dojde ke kolizi částice s nepohyblivou stěnou (rovnoměrný přímočarý pohyb částice), a



Obrázek 1.4: Algoritmus pro generování dat časového vývoje SFUM. Proměnná *timeToLive* říká, kolik kolizí částice s nepohybující se stěnou proběhne před konce programu; funkce *FindCollision* vrací časový okamžik nejbližší kolize částice s pohybující se stěnou vztažený k velikosti periody oscilace; funkce Derive vrací derivaci pohybové funkce pohybují se stěny v zadaném okamžiku

nastaví ji jako fázi t_0, dále nastaví rychlost při poslední kolizi částice s nepohyblivou stěnou v_0 na hodnotu v_1 a inkrementuje počítadlo kolizí částice s pohyblivou stěnou.

Složitost funkce pro výpočet dat časového vývoje SFUM pro n kolizí částice s nepohyblivou stěnou odpovídá asymptotické složitosti $O(n)$, neboť je v n krocích cyklu prováděn vždy nejméně dvakrát algoritmus pro nalezení kolize částice s pohyblivou stěnou, jehož asymptotická složitost odpovídá $O(k)$, kde pro k platí vztah (1.2), přičemž platí-li podmínky popsané výše, asymptotická složitost onoho algoritmu je konstantní, tudíž $O(1)$. Přesný počet provádění algoritmu pro nalezení kolize částice s pohyblivou stěnou nelze obecně určit, lze však říci, že vzhledem k podmínkám, že musí dojít vždy alespoň k jedné kolizi a od určité vzdálenost částice od nepohyblivé stěny už k žádné další kolizi dojít nemůže, bude počet provádění konečný.

Upravený algoritmus pro nalezení kolize částice s pohyblivou stěnou se od toho původního liší pouze o přidání kontroly vzdálenosti částice od nepohyblivé stěny. Pokud je částice k nepohyblivé stěně blíže, než může být pohyblivá stěna, hledání kolize se přeruší. Asymptotická složitost algoritmu se touto modifikací nemění, protože porovnávání je prováděno pouze jednou před každým průchodem cyklem.

1.3 Implementace návrhu

Pro implementaci algoritmu na výpočet časového vývoje SFUM byl zvolen programovací jazyk C, resp. C++. V úvahu zde byl brán požadavek na použití algoritmu v CAS *Wolfram Mathematica* a *Sage*, které oba podporují použití programů napsaných v jazyce C nebo C++ (viz analýza). Jazyk C/C++ je navíc nízkoúrovňový kompilovaný programovací jazyk, který nabízí vyšší výpočetní efektivitu, než jazyky interpretované, a vyšší míru přenositelnosti kódu, než implementace přímo pomocí příkazů v CAS, ovšem nižší, než interpretované jazyky.

Rozhraní sady funkcí pro výpočet dat časového vývoje SFUM implementuje funkce `float ** fermi(...)`, jejímž výstupem je reference na dynamicky alokované dvourozměrné pole o rozměrech $\lceil \frac{timeToLive}{granularity} \rceil \times 2$ (Význam parametru `granularity` viz níže). Rozhraní funkce je:

```
float ** fermi(
               float v,
               float t,
               int timeToLive,
               int granularity,
```

```
    int distance,
    float amplitude,
    int func
);
```

float v udává rychlost částice při poslední kolizi se stabilní stěnou.

float t udává časový okamžik v momentě poslední srážky částice s nepohyblivou stěnou, vztažený k periodě pohybové funkce pohybující se stěny.

int timeToLive udává počet odrazů od nepohyblivé stěny, pro něž mají být data vypočítána.

int granularity udává granularitu výsledků, tj. bude-li `granularity` rovna 1, na výstupu bude přesně `timeToLive` výsledků, bude-li `granularity` rovno 10, bude navýstupu přesně $\lceil \frac{timeToLive}{10} \rceil \times 2$ výsledků pro počet odrazů částice od nepohyblivé stěny rovný hodnotě parametru `timeToLive` s tím, že bude vždy devět výsledků zahozeno a uložen výsledek desátý.

int distance udává minimální vzdálenost pohyblivé stěny od nepohyblivé. Pohyblivá stěna se pohybuje v určité vzdálenosti od stěny nepohyblivé, v okamžiku, kdy dosáhla záporné amplytudy svého pohybu, se nachází nejblíže nepohyblivé stěny v rámci časového úseku jedné periody, jejich vzájemná vzdálenost je tedy minimální. Parametr `distance` udává právě tuto minimální vzdálenost.

float amplitude udává absolutní hodnotu amplitudy pohybu pohyblivé stěny. Pohybová funkce je definována výrazem distance + amplitude + amplitude $\cdot F(t)$, kde $F(t)$ je buďto funkce $\sin t$ nebo funkce „pila“ (viz obrázek 1.2) pohybující se od -1 do 1 s parametrem t.

int func je příznak nabývající hodnot 0 nebo 1 a udává, kterou periodickou funkci pohybová funkce pohyblivé stěny obsahuje. Hodnota 0 odpovídá funkci sinus; hodnota 1 odpovídá „pilové“ funkci.

Tato funkce implementuje algoritmus pro výpočet dat časového vývoje SFUM pro n kolizí částice s nepohyblivou stěnou a zároveň obsahuje implementaci rozšířeného algoritmu pro výpočet kolize částice s pohyblivou stěnou používaného k případnému nalezení druhotných kolizí částice s pohyblivou stěnou, neboť bylo zapotřebí měnit i některé další proměnné používané ve funkci `float ** fermi(...)`.

Zvlášť byl implementován algoritmus pro nalezení průsečíku dvou funkcí, a to pomocí funkce `float findPrimaryTCollision(...)`, která je volána v cyklu z funkce `float ** fermi(...)` za účelem nalezení fáze periody funkce pohybu pohyblivé stěny při první kolizi z částicí po odrazu částice od nepohyblivé stěny. Funkce vrací v návratové hodnotě reálné číslo typu `float` udávající fýzi periody pohybu pohyblivé stěny v okamžiku první vypočítané kolize s částicí.

Fází periody pohybu pohybující se stěny se zde rozumí časový okamžik v rámci časové periody, v případě funkce sinus je to hodnota v rozsahu 0 až 2π, v případě „pilové“ funkce v rozsahu 0 až 4. Dosazením tohoto časového údaje do pohybové funkce pohybující se stěny lze zjistit vzdálenost pohybující se stěny od statické stěny v dosazeném čase. Fáze pohybu pohybující se stěny je tedy časový údaj vztažený k periodě funkce zvolené hodnotou parametru `func`.

```
float findPrimaryTCollision(
                        float v0,
                        float t0,
                        float tStepSize,
                        float tFrontier,
                        float dToFrontier,
                        float a
                    );
```

float v0 udává rychlost částice v okamžiku její poslední kolize s nepohyblivou stěnou.

float t0 udává fázi periody pohybové funkce pohyblivé stěny v okamžiku poslední kolize částice s nepohyblivou stěnou.

float tStepSize je počáteční velikost časového kroku (viz návrh algoritmu).

float tFrontier udává fázi periody pohybové funkce pohyblivé stěny v okamžiku, kdy částice překoná po kolizi s nepohyblivou stěnou minimální vzdálenost mezi oběma stěnami. Od tohoto okamžiku může opět nastat kolize částice s pohyblivou stěnou.

float dToFrontier udává hodnotu pohybové funkce částice v okamžiku `tFrontier`.

float a udává absolutní hodnotu amplitudy periodické funkce obsažené v pohybové funkci pohyblivé stěny.

Dále program obsahuje několik dílčích funkcí, které implementují některé opakující se činnosti, nebo které umožňují odstranění dat. Mimo těchto funkcí obsahuje program také dvě globální proměnné, fungující jako příznaky. Jsou jimi:

int g_func říká, jakou periodickou funkci obsahuje předpis pohybové funkce pohyblivé stěny. Hodnota 0 odpovídá funkci sinus; hodnota 1 odpovídá „pilové“ funkci.

float g_period udává velikost periody pohybové funkce pohyblivé stěny. V případě funkce sinus je to hodnota $2 \cdot \pi$, v případě pilové funkce je to číslo 6.

Funkce `bool isEqual(float d1, float d2)` implementuje porovnávání dvou hodnot `d1` a `d2` v rámci dané tolerance. Využívána je pro porovnávání vzájemné pozice částice a pohyblivé stěny při výpočtu momentu kolize. V této implementaci je tolerance nastavena fixně na hodnotu $5 \cdot 10^{-5}$. Jsou-li hodnoty ekvivalentní v rámci dané tolerance, vrací funkce hodnotu `true`, jinak vrací hodnotu `false`.

d1 je první porovnávaná hodnota.

d2 je druhá porovnávaná hodnota.

Funkce `float intoRange(float x)` převede hodnotu předanou ve vstupním parametru `x` na odpovídající hodnotu v rozsah periody pohybové funkce pohyblivé stěny. Výslednou hodnotu vrací jako návratovou hodnotu. Parametr verb|x| je přirozené číslo, jenž je třeba převést na hodnotu v rámci periody pohybové funkce pohyblivé stěny.

Funkce `float saw(float x)` implementuje „pilovou“ funkci (viz obrázek 1.2). Výstupem je hodnota „pilové“ funkce v bodě `x`.

- Je-li hodnota `intoRange(x)` v rozmezí $\langle 0; 1 \rangle$, výsledkem je hodnota `intoRange(x)` (funkce je rostoucí).
- Je-li hodnota `intoRange(x)` v rozmezí $\langle 1; 3 \rangle$, výsledkem je hodnota $2 - $`intoRange(x)` (funkce je klesající)
- Je-li hodnota `intoRange(x)` v rozmezí $\langle 3; 4 \rangle$, je výsledkem hodnota `intoRange(x)`-4 (funkce je rostoucí).

x hodnota na časové ose, pro niž je hledána hodnota „pilové“ funkce.

Funkce `float sawDer(float x)` implementuje výpočet hodnoty první derivace „pilové“ funkce v bodě `x`.

- Je-li hodnota `intoRange(x)` v rozmezí $(0; 1)$, výsledkem je hodnota 1 (funkce je rostoucí).
- Je-li hodnota `intoRange(x)` v rozmezí $(1; 3)$, výsledkem je hodnota -1 (funkce je klesající)
- Je-li hodnota `intoRange(x)` v rozmezí $(3; 4)$, je výsledkem hodnota 1 (funkce je rostoucí).
- v bodech `intoRange(x)`$==$ 1 a `intoRange(x)`$==$ 3 nemá pilová funkce derivaci. Pro účely této implementaci tedy bylo definováno, že v těchto případech vrací funkce hodnotu 0. Hodnota 0 v této situaci má svůj fyzikální základ, neboť dojde-li ke kolizi částice s pohybující se svěnou v přesně okamžiku, kdy tato stěna dosáhla amplitudy svého pohybu, jeví se jako nepohybující se a částice se od ní pouze odrazí, aniž by se změnila velikost její rychlosti.

x hodnota na časové ose, pro niž je hledána hodnota první derivace „pilové“ funkce.

Funkce `float sinX(float x)` vypočítá sinus hodnoty z parametru `x`. Využívá k tomu funkci `float sin(float)` z knihovny `math.h`.

x hodnota na časové ose, pro niž je hledána hodnota funkce sinus.

Funkce `float sinXDer(float x)` vypočítá hodnotu první derivace funkce sinus v bodě předaném parametrem `x`. První derivace funkce $\sin x$ je $\cos x$, využita je proto funkce `float cos(float)` z knihovny `math.h`.

x hodnota na časové ose, pro niž je hledána hodnota první derivace funkce sinus v tomto bodě.

Funkce `float oscFunc(float x)` vrací hodnotu oscilační funkce obsažené v předpisu pohybové funkce pohyblivé stěny v bodě `x`. Oscilační funkce je vybrána pomocí příznaku v globální proměnné `int g_func` (hodnota 0 odpovídá funkci sinus; hodnota 1 odpovídá „pilové“ funkci).

x hodnota na časové ose, pro niž je hledána hodnota oscilační funkce.

Funkce `float oscDerFunc(float x)` vrací hodnotu první derivace oscilační funkce obsažené v předpisu pohybové funkce pohyblivé stěny v bodě `x`. Oscilační funkce je vybrána pomocí příznaku v globální proměnné `int g_func` (hodnota 0 odpovídá funkci sinus; hodnota 1 odpovídá „pilové“ funkci).

x hodnota na časové ose, pro niž je hledána hodnota první derivace oscilační funkce v tomto bodě.

Funkce `float initStepSize()` vrací inicializační velikost časového kroku použitého v algoritmu pro nalezení kolize částice a pohyblivé stěny. Aby platil vztah (1.2), vrací funkce konstatní hodnotu, a to hodnotu 1, která se jeví jako optimální pro užití ve spojitosti s funkcí sinus a „pilovou“ funkcí.

Funkce `void deleteFermiRes(float ** results, int hits, int gran)` uvolní paměť alokovanou pro dovourozměrné dynamické pole s výsledky.

float ** results je reference na pole výsledků o velikosti přesně $\lceil \frac{timeToLive}{2} \rceil \times 2$.

int hits je skutečný počet kolizí částice s nepohyblivou stěnou (nikoliv počet výsledků v poli).

int gran udává granularitu výsledků.

Kapitola 2

Analýza možností tvorby interaktivních webových prezentací pomocí Wolfram Mathematica

2.1 Wolfram Demonstrations Project

2.1.1 Wolfram CDF Player

Computable Document Format (CDF) je standard uvedený v roce 2011 skupinou *Wolfram Group* [29]. Nutnou podmínkou pro spuštění kódu je potřeba mít nainstalovaný *Wolfram CDF Player* [20]. Tento standard je určen pro tvorbu online dokumentů, které kromě běžného statického obsahu obsahují i prvky generované interaktivně uživatelem. Stránka vytvořená podle standardu CDF se tak stává určitým hybridem mezi obyčejným dokumentem a aplikací.

CDF dokumenty mohou být používány jednak k vytváření samostatných stránek zobrazených ve webovém prohlížeči, podporuje-li to platforma, jednak k vytváření vestavěných appletů do HTML stránek a v neposlední řadě také jako počítačové a mobilní aplikace. Dříve bylo možné do HTML stránek vložit CDF objekty pomocí tagu `<embed>`, a to následujícím způsobem:

```
<embed src="document.cdf" width="640" height="480">
```

Protože tento tag už není standardy *W3* podporován, ačkoliv většina webových prohlížečů jej ještě umí zobrazit, je vhodnější použít jej ve spojení s tagem `<object>`.

```
<object classid="clsid:612AB921-E294-41AA-8E98-87E7E057EF33"
 width="640" height="480"
 type="application/vnd.wolfram.cdf.text">
  <param name="src" value="document.cdf">
  <embed width="640" height="480"
   src="document.cdf type="application/vnd.wolfram.cdf.text">
</object>
```

Pro větší flexibilitu je však doporučeno použít *Wolfram CDF Embed Script*, což je open source knihovna v jazyce JavaScript. Tato knihovna už nevyžaduje vkládání žádných jiných knihoven a garantuje kompatibilitu ve všech prohlížečích, dále poskytuje způsob, jak zjistit, zda je v uživatelově prohlížeči nainstalován CDF modul. V případě, že tento modul chybí, zobrazí logo *Wolfram CDF Player* a odkaz na jeho stažení, knihovna zároveň nabízí možnost zobrazení statických obrázků, pokud chybí modul CDF. *Wolfram CDF Embed Script* se pro vložení CDF objektu použije následujícím způsobem:

```
<script type="text/javascript"
        src="http://www.wolfram.com/
             cdf-player/plugin/v2.1/cdfplugin.js">
</script>
<script type="text/javascript">
        var cdf=new cdfplugin();
        var width = 640;
        var height = 480;
        cdf.embed('/cesta/k/souboru.cdf', width, height);
</script>
```

Wolfram CDF Player je program od firmy *Wolfram*, který umožňuje přehrávat soubory formátu CDF. Zjednodušeně řečeno jde o upravenou verzi programu Mathematica, odlehčenou o některé nepotřebné součásti a o příklady. Kvůli tomu se velikost jeho instalačního souboru pohybuje okolo 626,3 MB (verze 9.0.1), samotná instance programu potom zabírá cca 500 MB.

Program Wolfram CDF Palyer umožňuje zobrazovat a ilustrovat *Demonstrace* přímo v prohlížeči, rovněž umožňuje interakci se všemi vstupy a ovládacími prvky a manipulaci s 2D a 3D grafickými elementy v *Demonstracích*. *Wolfram CDF Player* lze nainstalovat v první řadě jako doplněk do webového prohlížeče na platformách Windows a Mac OS X, pro Linux není v této podobě k dispozici. Dále jej lze používat jako samostatnou desktop aplikaci na platformách Windows, Mac OS X a Linux. Bohužel zatím není k dispozici žádná verze pro mobilní zařízení, což může být zapří-

činěno mimo jiné i velikostí tohoto programu. Stažení, instalace a používání programu *Wolfram CDF Player* jsou bezplatné, uživatel pouze vybere účel používání a zadá svůj email.

2.1.2 Demonstrace

Wolfram Demonstrations Project umožňuje registrovaným uživatelům vytvářet a zveřejňovat interaktivní prezentace zvané *Demonstrace*, vytvořené v CAS *Wolfram Mathematica*. *Demonstrace* si pak může prohlížet a stáhnout kdokoliv bez nutnosti přihlášení. Registrace uživatelů je nutná k publikaci *Demonstrací* a je bezplatná, stejně tak i nahrávání a zveřejnění jednotlivých *Demonstrací*. Pro přehrávání *Demonstrací* se používá program *Wolfram CDF Player*, uživatel tedy pro interakci s *Demonstracemi* nepotřebuje software *Mathematica*. *Demonstraci* je možno vkládat do vlastních webových stránek pomocí kódu v jazyce JavaScript, který je generován na stránce *Demonstrace* v sekci *Share* (Sdílet).

Demonstrace je mini-aplikace vytvořená ve standardní kopii softwaru *Mathematica* bez nutnosti instalace dalších součástí. Autor *Demonstrace* vytvoří v programu *Mathematica* nový soubor typu *Demonstrace*, ten hned po vytvoření obsahuje šablonu *Demonstrace* a pomocí funkce `Manipulate` [27] autor vytvoří interaktivní rozhraní dynamicky zobrazující výsledky zadané funkce, soubor s tímto rozhraním pak nahraje na web projektu na stránce *Upload* (Nahrát) a vyplní autorské informace. Server pak tuto šablonu převede do formátu CDF. Po nahrání na server projektu je *Demonstrace* podrobena recenzování a testování ze strany demonstrations.wolfram.com.

Šablona pro *Wolfram Demonstrations Project* se vytvoří v programu *Wolfram Mathematica* přes `File`→`New`→`Demonstration` a je rozdělena do několika částí, pomineme-li nadpis – název demonstrace, jsou to následující povinné:

Inicializační kód (Initialization Code) – do této části šablony je třeba umístit jakýkoliv kód, který je třeba vyhodnotit ještě před vyhodnocováním příkazu `Manipulate`. Obsah této sekce je ve finálním vygenerovaném notebooku automaticky vestavěn do příkazu `Manipulate`. Ovšem pro používání balíčků je vhodné použít přímo ve funkci Manipulate volbu `Inicialization` s `Get` nebo `Need`. Pokud je inicializační kód používán, je třeba použít v `Manipulate` volbu `SaveDefinitions` → `True`.

Manipulate – do této sekce je vkládán přímo kód příkazu `Manipulate` a následně je vyhodnocen. Jeho výsledkem bude konkrétní interaktivní demonstrace. Popisky jednotlivých ovládacích prvků, jako tlačítka,

posuvníky textová pole atd., by měly být co nejkonkrétnější, ovšem pokud je to nezbytné, je možné jejich význam rozvést v další části šablony.

Popisek (Caption) – do této části šablony nepatří kód, ani grafika, či obrázky, nýbrž obyčejný text. Ten by měl být v rozmezí tří až pěti vět a mělo by z něj být možné *Demonstraci* porozumět. V sekci by rovněž nemělo být příliš detailů, ani reference, pro ně bude prostor dále.

Náhled (Thumbnail) – pro vytvoření náhledu stačí kopírovat výstup sekce *Manipulate* a vložit jej do této sekce. Náhled by měl pokud možno ukazovat výstup ve vizuálně co možná nejzajímavější podobě, před kopírováním a vložením náhledu je tedy vhodné pomocí manipulovatelných prvků tento výstup připravit. V žádném případě není povoleno převádět výstup `Manipulate` na bitmapu.

Snímky (Snapshots) – do této sekce se vkládají alespoň tři další kopie výstupů příkazu `Manipulate`. Tyto výstupy by měly mít rozdílná rozestavení ovládacích prvků, aby ukázaly celou škálu možností *Demostrace*. Snímky popisující chování a ovládání *Demonstrace* do této sekce nepatří, pro ně je vyhrazeno místo níže. Převádět snímky na bitmapu je zakázáno.

Autorství (Authoring Information) – v této sekci má být napsáno jméno autora, případně autorů *Demonstrace*. Pokud *Demonstrace* vychází z jiného kódu, v této sekci by tento fakt měl být uveden.

Šablona obsahuje rovněž několik nepovinných sekcí:

Detaily (Details) tato sekce smí obsahovat pouze text, žádnou grafiku a obrázky, ani kód. Zde by měla být aplikace popsána detailněji, než v sekci Popisek, ovšem stále by tento popis měl být co nejstručnější. Sekce rovněž může obsahovat dodatečné vysvětlení funkce jednotlivých ovládacích prvků a chování *Demonstrace*. V neposlední řadě je možno zde umístit popis obrázků v sekcích Snímky a Náhled, či definovat symboly, které doposud nebyly definovány. Do této sekce je rovněž vhodné vložit reference na díla jiných autorů, pokud byla použita, či citována.

Pokyny ovládání (Control Suggestions) tato sekce obsahuje několik zaškrtávacích políček, pomocí jejichž zaškrtnutí autor Demonstrace určí, které operace bude možno s výstupem příkazu `Manipulate` provádet. V seznamu (*Mathematica 9*) se nachází následující políčka:

- Resize Images (Změna velikosti obrázku)
- Rotate and Zoom in 3D (Otáčení a přiblížení/oddálení ve 3D grafice)
- Drag Locators (Manipulace uchopením kurzorem myši s prvky `Locator` v grafu)
- Create and Delete Locators (Vytvoření a mazání prvku Locator v grafu)
- Slider Zoom (Umožnění používání kláves `Alt`, resp. `Alt+Ctrl+Shift` k zadání přesnějších hodnot pomocí posuvníků)
- Gamepad Controls (Povolit/Zakázat ovládání *Demonstrace* dalšími ovládacími vstupními zařízeními připojenými k počítači)
- Automatic Animation (Automatická animace)
- Bookmark Animation (Předpřipravená animace)

Vyhledávané pojmy (Search Terms) – do této sekce se vkládají slova nebo slovní spojení, která by mohli lidé vyhledávat ve spojitosti s *Demostrací*. Každé slovo nebo slovní spojení musí být vloženo do samostatné buňky bez používání velkých písmen, pokud se nejedná o vlastní jména. Sekce by neměla obsahovat kategorizaci *Demonstrace*, ta je přidávána později, ale synonyma, výrazy odkazující na specifické části demonstrace atp.

Související odkazy (Related Links) – tato sekce obsahuje odkazy na jiné stránky na wolfram.com, se kterými má tato *Demonstrace* nějakou spojitost. Odkazy mimo server wolfram.com patří do sekce Detaily.

2.2 Programování ve Wolfram Mathematica

Wolfram Mathematica je velice komplexní matematický software. Uživateli umožňuje mimo běžných výpočetních operací, jako je řešení soustav rovnic a vykreslování grafů, také definovat vlastní funkce a procedurálně programovat. Rovněž je možné přímo kompilovat zdrojové kódy v jazyce C ve funkci `CreateExecutable` [21], instalovat již kompilované moduly funkcí `Install` [23] a převádět příkazy z *Mathematica* na zdrojový kód jazyka C (funkce `CForm`), Fortran (funkce `FortranForm`) a na soubory jiných podporovaných formátou (funkce `Export`). Pro dosažení cílů práce je zapotřebí pouze podpora procedurálního programování a užití zdrojového kódu v jazyce C.

V *Mathematica* je procedurální programování realizováno pomocí specifických funkcí, jako jsou například `Function`, `For`, `If` a další. Nejedná se tedy o klasické programování jako v jazycích C, Pascal nebo Java, či psaní skriptů například v jazyce AWK, ale o používání specifických funkcí zabudovaných v softwaru. Z tohoto důvodu může být někdy výpočet triviálních úloh trvat déle, než kdyby k této činnosti byly použity programy napsané například v jazyce C.

Funkce `CreateExecutable` slouží ke kompilaci zadaného zdrojového kódu jazyca C a vytvoření spustitelného souboru z tohoto kódu. Jako první parametr přijímá tato funkce řetězec, nebo seznam řetězců, či jméno souboru, nebo seznam jmen souborů se zdrojovým kódem. Jako druhý parametr přijímá řetězec, který reprezentuje název souboru a zároveň jméno, přes které bude program ve spustitelném souboru dostupný. Pro správné fungování funkce `CreateExecutable` je potřeba ji nejprve načíst pomocí příkazu `Needs["CCompilerDriver‘"]`. Funkce `CreateExecutable` se spouští při každém volání vyhodnocení *Mathematica notebooku* [22], či jeho příslušné části, kompilace tedy proběhne pokaždé znovu.

Naproti tomu funkce `Install` načte již zkompilovaný program a vytvoří odkaz, pomocí kterého jej lze spouštět. Tento program lze posléze odebrat voláním funkce `Uninstall`. Funkce `Install` však umí pracovat pouze s programy opatřenými šablonou `.tm` a používající protokol *MathLink*.

2.3 Využití kompilovaného kódu za pomoci protokolu *MathLink*

MathLink je komunikační protokol [18] zajišťující přijímání a odesílání výrazů systému *Mathematica* mezi procesy, součástí *Wolfram Mathematica* je od verze 2.0 vydané v roce 2001. Jednak umožňuje komunikaci mezi dvěma různými *Mathematica notebooky* [25] a jednak komunikaci mezi *notebookem Mathematica* a funkcemi externího programu v jazyce C [28]. Podporuje také síťovou komunikaci mezi výše zmíněnými.

Aby mohly být funkce z externích programů z programu *Mathematica* volány [24], musí být jednak tyto programy s *Mathematica* spojeny, což se provede pomocí vestavěné funkce `Install`, a jednak musí připojovaný program splňovat pro používání protokolu *MathLink*.

První podmínkou je, že do programu musí být vložen hlavičkový soubor `mathlink.h`, dodávaného v *MathLink DeveloperKit* spolu s programem *Wolfram Mathematica*. Tento hlavičkový soubor obsahuje deklarace funkcí a datových typů používaných pro komunikaci s *Mathematica notebookem*

přes protokol *MathLink*

Druhou podmínkou je nutnost, aby funkce `main` dodržela předepsanou podobu:

```
int main(int argc, char *argv[]) {
  return MLMain(argc, argv);
}
```

`MLMain(int argc, char ** argv)` je funkce spouštějící komunikační spojení mezi externím programem a *notebookem Mathematica* skrze volání funkce `Install` programu *Mathematica*. Funkce nejdříve otevře spojení přes protokol *MathLink* s parametry předávanými v `argv` a poté přejde do smyčky, kde čeká, až od *Mathematica notebooku* přijme objekt typu `CallPacket`, což je paket zapouzdřující požadavek na vyvolání externí funkce reprezentovaný hodnotou typu `integer` a seznam obsahující argumenty.

Kód funkce `MLMain(int argc, char ** argv)` je automaticky vytvářen příkazem `mprep`, který ze souboru šablony `.tm` generuje kód jazyka C obsahující funkce a další prvky nezbytné pro spojení externího programu s *Mathematica notebookem* pomocí protokolu *MathLink*, nebo příkazem `mcc`, který slouží pro překlad kódu jazyka C a C++ na spustitelný program, jenž může být posléze použit jako externí funkce volaná z *Mathematica notebooku* pomocí funkce `Install` programu *Mathematica*.

Třetí podmínkou je, že program, který má být volán z *Mathematica notebooku* musí být kompilován s odpovídající šablonou `.tm`. Tato šablona obsahuje popis funkce nebo funkcí v externím programu, které mohou být volány z *Mathematica notebooku*. Program s funkcí pro externí volání z *Mathematica notebooku* musí být překládán i s kódem generovaným z této šablony a to buďto manuálně příkazem `mprep`, nebo automaticky dodáním `.tm` souboru jako argumentu spolu se zdrojovými kódy programu v jazyce C nebo C++ funkci `mcc`, v níž je před samotným překladem ze šablony `.tm` příkazem `mprep`, kterému je jako argument předán dodaný název souboru s příponou `.tm`, vygenerován kód v jazyce C.

Šablona `.tm` tedy slouží k definici rozhraní mezi Mathematica a externím programem. Skládá se z několika povinných částí:

- `:Begin:` označuje začátek šablony;
- `:Function: name` řetězec `name` je název volané funkce ve zdrojovém kódu programu;
- `:Pattern: Function {[]argument\_ArgumentType,{]}` řetězec `Function` je jméno, jehož prostřednictvím bude moci být funkce z externího programu volána v *Mathematica* `argument` označuje jméno

argumentu funkce a `ArgumentType` oddělený od jména argumentu znakem „_“ označuje datový typ argumentu(viz tabulka);

- `:Arguments:` a následuje seznam argumentů bez jejich datových typů ve formátu Mathematica výrazu;
- `:ArgumentsTypes:` následuje seznam datových typů argumentů funkce v C programu, seznam je ve formátu *Mathematica* výrazu;
- `:ReturnType:` jenž udává návratový typ funkce z hlediska softwaru Mathematica, tímto typem může být buďto některý z tabulky datových typů, nebo typ `Manual` který značí, že návratový typ bude nastaven až ve funkci externího programu;
- `:End:`, označující konec šablony.

Dále může obsahovat i nepovinné prvky:

- `:Evaluate: expression`, kde řetězec `expression` reprezentuje výraz ve formátu *Mathematica*, který se vyhodnotí při instalaci funkce (tedy při volání funkce `Install` z *notebooku Mathematica*); zde lze nastavit také text, který bude zobrazen po zadání příkazu `? MyFunction` ve *Wolfram Mathematica*. `MyFunction` reprezentuje název importované funkce, řetězec `popis` pak odpovídá textu, který bude vypsán na výstup po zadání výše uvedeného příkazu.

  ```
  :Evaluate: MyFunction::usage="popis"
  ```

- `::comment`, kde řetězec `comment` je libovolný řetězec znaků; jakýkoliv text za znaky `::` až do konce řádku je považován za komentář.

MathLink podporuje přenos fatových formátů uvedených v tabulce 2.1.

Čtvrtou podmínkou je, aby funkce externího programu volaná z *Mathematica notebooku* buďto předávala jako návratovou hodnotu takový datový typ, jaký je uveden v šabloně, nebo, je-li v šabloně uveden návratový typ jako `Manual`, aby byl definován ve zdrojovém kódu funkce a návratová data odeslána pomocí funkcí z `mathlink.h` a funkce nevracela žádnou hodnotu.

Po sestavení a přeložení kódu je pak tedy vytvořen spustitelný soubor, který na základě předávaných argumentů komunikuje s protistranou pomocí protokolu *MathLink*. Tento program lze k *Mathematica notebooku* připojit funkcí `Install[program]`, kde `program` je buďto absolutní cesta k danému souboru, nebo relativní cesta vzhledem k aktuálně nastavenému výchozímu adresáři, kterou lze nastavit funkcí `SetDirectory`. Další užitečná funkce

Tabulka 2.1: Datové typy podporované protokolem MathLink použitelné jako datové typy argumenty a návratových hodnot v .tm sabloně

Wolfram Mathematica	**C/C++, mathlink.h ekvivalenty**
`Integer16`	`short`
`Integer32`	`int`
`Integer64`	`mlint64`
`Real32`	`float`
`Real64`	`double`
`Real128`	`mlextended_double`
`Integer16List`	`short*`, `int` (délka seznamu)
`Integer32List`	`int*`, `int` (délka seznamu)
`Integer64List`	`mlint64*`, `int` (délka seznamu)
`Real32List`	`float*`, `int` (délka seznamu)
`Real64List`	`double*`, `int` (délka seznamu)
`Real128List`	`mlextended_double*`, `int` (délka seznamu)
`String`	`char*`
`ByteString`	`unsigned char*` a `int` (délka řetězce)
`UCS2String`	`unsigned short*`
`UTF8String`	`unsigned char*`, `int` (počet bajtů řetězce), `int` (characters)
`UTF16String`	`unsigned short*`, `int` (délka řetězce), `int` (characters)
`UTF32String`	`unsigned int*`, a `int` (délka řetězce)
`Symbol`	char*
`ByteSymbol`	`unsigned char*`, `int` (délka řetězce)
`UCS2Symbol`	`unsigned short*`, `int` (délka řetězce)
`UTF8Symbol`	`unsigned char*`, `int` (počet bajtů řetězce), `int` (characters)
`UTF16Symbol`	`unsigned short*`, `int` (délka řetězce), int (characters)
`UTF32Symbol`	`unsigned int*`, `int` (délka řetězce)
`ThisLink`	`MLINK`
`\$CurrentLink`	`MLINK`
`Manual`	`void`

v tomto kontextu je `NotebookDirectory[]` vracející absolutní adresu adresáře, kde je uložen aktuální *Mathematica notebook*. Funkce `Install` vrací jako návratovou hodnotu odkaz na připojenou externí funkci, pomocí kterého lze s touto funkcí komunikovat.

2.3.1 Příklad programu

Máme externí funkci `multipletwo` v jazyce C, která vynásobí dvě přirozená čísla typu `double` a chceme ji použít v programu *Wolfram Mathematica* voláním `MultipleTwo`.

Vytvoříme tedy následující soubory:

- *multiple.c*, který obsahuje zdrojový kód funkce `multipletwo`.

```
# multiple.c
#include "mathlink.h"

extern double multipletwo( double i, double j);

double multipletwo( double i, double j)
{
  return i * j;
}

int main(int argc, char* argv[])
{
  return MLMain(argc, argv);
}
```

- *multipletwo.tm*, který obsahuje `.tm` šablonu.

```
doulbe multipletwo P(( double, double));

:Begin:
:Function:       multipletwo
:Pattern:        MultipleTwo[i_Double64, j_Double64]
:Arguments:      { i, j }
:ArgumentTypes:  { Double64, Double64 }
:ReturnType:     Double64
:End:
```

```
:Evaluate: MultipleTwo::usage = "MultipleTwo[x, y]
gives the multiplication of two doubles x and y."
```

- *Makefile* soubor, pomocí něho příkazem `make` kód přeložíme.

```
VERSION=9.0
MLINKDIR = /usr/local/Wolfram/Mathematica/\
           9.0/SystemFiles/Links/MathLink/\
           DeveloperKit/Linux
SYS = Linux
CADDSDIR = ${MLINKDIR}/CompilerAdditions
EXTRA_CFLAGS=-m32

INCDIR = ${CADDSDIR}
LIBDIR = ${CADDSDIR}

MPREP = ${CADDSDIR}/mprep
RM = rm

CC = /usr/bin/gcc
CXX = /usr/bin/c++

BINARIES = multiple

all : $(BINARIES)

multiple : multipletm.o multiple.o
${CXX} ${EXTRA_CFLAGS}\
-I${INCDIR} multipletm.o multiple.o\
-L${LIBDIR} -lML32i3 -lm -lpthread -lrt -lstdc++ -o $@

multipletm.cpp : multiple.tm
${MPREP} $? -o $@

clean :
@ ${RM} -rf *.o *tm.cpp $(BINARIES)
```

Když máme vytvořeny tyto soubory, spustíme příkaz `make` a přeložíme tak program. Výstupem bude soubor `multipletwo`, jenž musíme nastavit jako spustitelný. Nakonec v *notebooku* programu *Wolfram Mathematica*

napíšeme příkaz `link = Install["/cesta/k/miultipletwo"]` a tak můžeme pomocí příkazu `MultipleTwo` přistupovat k funkci `multipletwo` ze souboru `multipletos.c`, např.: `MultipleTwo[1.5,2.7]`.

2.4 Tvorba interaktivních rozhraní ve Wolfram Mathematica

2.4.1 Manipulate

V softwaru *Wolfram Mathematica* je možné kromě statické grafiky vytvářet i interaktivní rozhraní, které uživateli umožní ovlivnit vykreslovaná data v reálném čase, aniž by musel zasahovat přímo do kódu v *Mathematica notebooku*. K tomuto slouží příkaz `Manipulate`.

Použitím příkazu `Manipulate` lze vytvořit poměrně širokou škálu interaktivních mini-aplikací za pomoci několika málo řádků kódu, aniž by se uživatel musel učit novým komplikovaným konceptům anebo principům tvorby uživatelských rozhraní. Výstupem vyhodnocení příkazu `Manipulate` je objekt připomínající applet, video, nebo jiný speciální modul. Jak již bylo zmíněno v úvodu, nejedná se pouze o statické zobrazení vypočítaných dat, ale objekt s interaktivním uživatelským rozhraním. Toto rozhraní umožňuje pomocí ovládacích prvků (např. posuvníky, tlačítka atd.) ovlivnit vykreslovaný výstup přes parametry, jejichž hodnota se ovládacími prvky mění. Nový obsah pole zobrazujícího výstup se pak generuje při každé změně stavu některého z ovládacích prvků, aniž by bylo třeba opětovného vyhodnocení buňky s příkazem `Manipulate`. Tento příkaz rovněž umožňuje vytvořit z výstupního interaktivního objektu automatizovanou animaci zobrazující konkrétní sadu výstupů vzhledem ke změně daných parametrů.

Výstupem funkce `Manipulate` je, jak již bylo zmíněno, speciální objekt – interaktivní rozhraní. Toto rozhraní je vizuálně rozděleno do dvou částí – ovládacích prvků a pole pro výstup. Vzájemná pozice a tvar těchto částí může být ovlivněna parametry příkazu `Manipulate`. Blok (nebo bloky) ovládacích prvků obsahuje všechny posuvníky, tlačítka, oddělovače, nadpisy, popisky ovladačů a další volitelné součásti přesně v pořadí, ve kterém jsou předávány jako parametry příkazu `Manipulate`. Pole pro výstup je zpravidla bílým obdélníkem, do nějž jsou vypisovány výstupy ve formátu shodném s běžným výstupem vyhodnocení příkazů v buňce *notebooku* programu *Wolfram Mathematica*.

Příkaz `Manipulate` má syntaxi:

```
Manipulate[expression,{manipulator settings},...]
```

Parametr `expression` zastupuje libovolnou posloupnost příkazů v programu *Wolfram Mathematica*, v nichž je možno pracovat s interaktivně ovládanými parametry předávanými v druhém parametru. Příkazy obsažené v parametru `expression` se vyhodnocují po každé změně některého ovládacího prvku (tedy při každé změně parametru) jejich výstup je potom vypsán do pole pro výstup rozhraní výsledného objektu funkce `Matupulate`. Dalšími parametry (`{manipulator settings}`) jsou ovládací prvky aplikace. Zde existuje několik variant, jak tyto prvky definovat:

1. `{u,...}` vytváří ovládací prvek umožňující manipulovat s hodnotou parametru `u`.
2. `{u, umin, umax}` vytváří ovládací prvek umožňující manipulovat s hodnotou parametru `u` v intervalu od `umin` včetně do `umax` včetně.
3. `{u, umin, umax, du}` vytváří ovládací prvek umožňující manipulovat s hodnotou parametru `u` v intervalu od `umin` včetně do `umax` včetně po krocích délky `du`.
4. `{u, {u1, u2,...}}` vytváří ovládací prvek umožňující přidělovat parametru `u` jednu hodnotu ze seznamu `{u1, u2, ...}`.
5. `{{u, uinit}, umin, umax,...}` vytváří ovládací prvek umožňující manipulovat s hodnotou parametru `u` v intervalu od `umin` včetně do `umax` včetně s implicitní hodnotou nastavenou na `uinit`.
6. `{{u, uinit, ulbl},...}` vytváří ovládací prvek umožňující manipulovat s hodnotou parametru `u` s implicitní hodnotou nastavenou na `uinit` a popiskem `ulbl`.
7. `"cu"->{u,...}` připojuje parametr u k externímu ovládacímu zařízení reprezentovanému odkazem `"cu"`.

Jako implicitní ovládací prvek je pro jednotlivé parametry nastaven posuvník. V obsahu popisu proměnného parametru však lez nastavit, že hodnotu parametru bude možno měnit pomocí 2D posuvníku, což je manipulovatelný bod v soustavě dvourozměrných souřadnic, dále pomocí tlačítka, jemuž je přidělena konkrétní hodnota, pomocí rozbalovacího seznamu, pomocí zaškrtávacích políček vracejících pro daný parametr hodnoty `True`, nebo `False`, pomocí polohovatelného bodu v grafickém výstupu funkce `Manipulate`, nebo pomocí textového pole. Použít lze také zvláštní ovládací prvky, jimiž jsou horizontální oddělovač, přímý text, buňka (rozuměno buňka *Mathematica notebooku*), náhled, animace a další, a které přímo neovlivňují žádné parametry vyhodnocovaného výrazu.

2.4.2 Grafické zpracování dat

Pro vizuální prezentaci většího množství vypočtených hodnot lez ve *Wolfram Mathematica* použít zpravidla dvou grafických funkcí, jsou jimi `ListPlot` [26] a `ArrayPlot` [19]. Zatímco `ListPlot` vyžaduje na vstupu pole hodnot, nebo pole dvourozměrných souřadnic, které zanese do grafu jako body, `ArrayPlot` pracuje s maticemi `m x n`, v nichž je jako hodnota na daných souřadnicích uvedena barva bodu, který bude vykreslen. *Wolfram Mathematica* samozřejmě disponuje širokou škálou dalších možností, jak vizualizovat výsledky, ovšem pro demonstraci *Fermiho akcelerace* jsou vhodné právě výše zmíněné funkce, neboť pouze ty umožňují generování grafů z pole hodnot.

Funkce `ListPlot` vrací jako výstup svého vyhodnocení dvourozměrný grafický prvek (obrázek) s vizualizací bodů zadaných jako vstup. Je takto možné vizualizovat jednak hodnoty jednorozměrného pole, kdy jsou hodnoty z pole vynášeny na osu y a na ose x se postupuje krokově, kdy každé další hodnotě z vstupního pole je přiřazena hodnota na ose x o velikost kroku vyšší, než té předchozí. Implicitní délka kroku je přitom nastavena na hodnotu 1. Jednak také je možno vizualizovat dvourozměrné pole, a to buďto pole souřadnic, tedy pole dvojic hodnot, či dvourozměrných vektorů, kde první hodnota odpovídá hodnotě na ose x a druhá hodnota odpovídá hodnotě na ose y, nebo takové dvourozměrné pole, které je vlastně seznamem seznamů hodnot. V takovém případě se hodnoty na ose x přidělují obdobně jako v případě jednorozměrného pole. Poslední variantou dvourozměrného pole, které funkce `ListPlot` umožňuje, je tabulka o dvou sloupcích, kde hodnota v prvním sloupci (klíči tabulky) je hodnotou na ose x a hodnota v druhém sloupci tabulky je hodnotou na ose y.

Funkce `ArrayPlot` přijímá jako vstupní parametr seznam dvojic souřadnic s přiřazenou hodnotou, kterou může být buďto sytost barvy, pakliže se jedná o vykreslování ve stupnici jedné barvy, nebo RGB barva (objekt `Color`). Z tohoto seznamu pak funkce sestaví nespojitý graf barevných čtverců. Čtverce jsou umístěny v síti a vybarvovány podle souřadnic předávaných ve vstupním parametru, přičemž každá tato dvojice souřadnic říká, v kolikátém řádku a sloupci se vykreslovaný čtverec nalézá.

Pro vynášení velkého množství bodů (dvojic souřadnic x, y) do relativně malého grafu se jeví jako vhodnější použít funkce `ArrayPlot` v kombinaci s mapovací funkcí. Funkce `ArrayPlot` totiž ve výsledku vykresluje jen tolik bodů, kolika je skutečně přidělena nějaká barevná hodnota. Je-li tedy síť čtverců grafu dostatečně hustá, například tak, že počet řádků odpovídá výšce grafu v pixelech a počet sloupců odpovídá šířce grafu v pixelech, pracuje funkce `ArrayPlot` při velkém množství vizualizovaných dat s relativně menší maticí, než funkce `ListPlot`, je tedy paměťově méně náročná.

Kapitola 3

Analýza možnosti tvorby interaktivních webových prezentací pomocí Sage

3.1 Seznámení se Sage

Sage (někdy také *Sagemath*) je Open Source [12] CAS s podporou pro platformy Linux, Solaris, Mac OS X a pomocí virtualizace i pro Windows. Díky otevřené licenci je *Sage* zdarma, ovšem jeho užívání nemusí být natolik komfortní, jako u některých komerčních produktů; dostupný je na webu `www.sagemath.org/`.

Sage slouží k výpočtům složitých i jednoduchých matematických operací a k matematickému a geometrickému experimentování. Systém používá programovací jazyk Python a podporuje procedurální, funkcionální a objektově orientované programování. Mimo skriptů v Pythonu obsahuje také kompilátory jazyka C a používá jazyk *Cython* [6], což je upravená verze Pythonu, která je před interpretací převedena na kód v jazyce C a posléze zkompilována kompilátorem jazyka C.

Sage je vyvíjen od roku 2005, lídrem projektu je profesor matematiky William Stein z University of Washington v Seattle, Washington, USA. *Sage* si za několik málo let své existence vytvořil uživatelskou komunitu, která systém dále rozvíjí, avšak vzhledem k faktu, že *Sage* je Open Source projekt pod licencí GNU, vývojáři musejí využívat pouze otevřených zdrojů. Od verze 3.0 vydané v roce 2008 je součástí *Sage* příkaz `interact`.

Sage má dvě uživatelská rozhraní – konzolové a webové; své vlastní GUI *Sage* nemá. Zásadní nevýhodou je, že webové rozhraní lze spustit a ukončit pouze z konzolového rozhraní. Textové rozhraní je vhodné pro výpočty,

které nevyžadují delší zápis, ovšem pro psaní delších skriptů se výhodnější rozhraní webové. ve webovém rozhraní si uživatel nejdříve vytvoří nový *worksheet* [14], nebo si zvolí již existující a edituje ho, případně s ním jinak pracuje.

3.2 Interaktivní webové rozhraní v Sage

3.2.1 Sage Interact

Sage Interact Website je online kolekce interaktivních aplikací [13] v *Sage* používajících příkaz `interact` utvářená uživatelskou komunitou *Sage Interact*. Uživatelé vytvářejí interaktivní aplikace přímo na své stránce po přihlášení. Na stránce pro vytvoření interaktivní aplikace uživatel vyplní formulář pro informace o aplikaci a vloží kód. Přihlásit se lze přes uživatelské účty na službách *Sage Interact Website*, *Google*, *Yahoo!*, *AOL*, *MyOpenID*, *OpenID*, *LiveJournal*, *Worldpress*, *Blogger*, *Verisign*, *ClaimID*, *ClickPass* a *Google Profile*. Veškeré výpočty spojené s vyhodnocováním interaktivní aplikace probíhají přímo na serveru a není třeba instalovat do počítače uživatele žádný další software.

Na stránce vložení nové interaktivní aplikace vyplní autor následující údaje:

Title (povinná položka) – Název aplikace

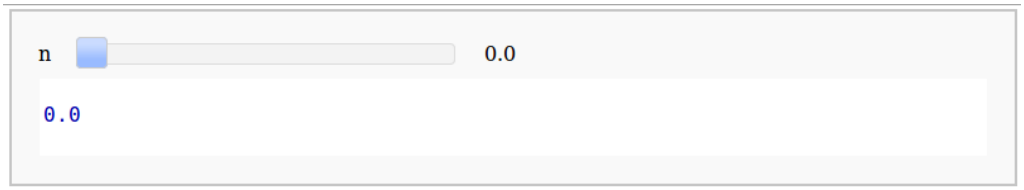
Code (povinná položka) – *Sage* kód k vyhodnocení. Toto pole odpovídá jedné buňce webového rozhraní *Sage Cell Server* [11], přímo na stránce je možné provést vyhodnocení kódu stiskem tlačítka *Evaluate* (vyhodnotit), jímž lze zároveň zkontrolovat funkčnost vloženého kódu.

Summary – Souhrn, nebo také abstrakt, který bude zobrazen na stránce s ukázkami.

Description (povinná položka) – Toto je textové pole podporující formátování textu. Autor vyplní popis činnosti vloženého kódu. Zároveň

Tags – klíčová slova

Images – Jakékoliv přidané screenshoty a obrázky, které se zobrazí na stránce s ukázkami a poté na stránce aplikace. Soubory musí být menší, než 2 MB a povoleny jsou jen formáty png, gif, jpg/jpeg a rozlišení menší, než 1024×1024.



Obrázek 3.1: Ukázka použití příkazu *interact*

Files – Nahrání souborů souvisejících s kódem, limit velikosti souboru je 2 MB a povoleny jsou formáty txt sws pdf py rst.

Author Credit – Do tohoto pole autor vyplní informace o autorství třetích stran, je-li to třeba.

Revision information – Informace o provedených změnách a jejich důvodu.

Před uložením údajů je možné zobrazit náhled.

Příkaz `interact` v *Sage* funguje tak, že vytvoříme třídu, jíž v parametrech konstruktoru předáme informace o proměnných, kterým budou přiděleny ovládací prvky, nebo funkci s parametry, jež bude možno měnit pomocí ovládacích prvků. Při každé změně hodnoty některého z těchto parametrů se vykoná celý kód za příkazem `interact`. Příkladem takové funkce je následující funkce `f` pro výpočet druhé mocniny čísel od 0 do 10, nodnota parametru `n` je zadávána interaktivním ovládacím prvkem, implicitně posuvníkem. Náhled vyhodnocení následujícího kódu viz obrázek 3.1.

```
@interact
def f(n=(0,10)):
    print n^2
```

Hodnoty proměnných ve vyhodnocovaném kódu je možné měnit pouze textovými poli, posuvníky, tlačítky a rozbalovací nabídkou. K dispozici není sbírání souřadnic kliknutím do vygenerovaného grafu, neboť se nejedná o interaktivní objekt, ale pouze o grafiku.

Sage Cell Server je open source webové rozhraní pro *Sage*, které může být navíc použito k vkládání výpočetních modulů do jakýchkoliv webových stránek. K použití při vkládání výpočetních modulů do vlastních webových stránek lze *Sage Cell Server* provozovat na vlastním serveru [15] přístupném z internetu, nebo využít instanci veřejně dostupnou na servru *Sagemath* `http://sagecell.sagemath.org/`.

Postup vložení výpočetního modulu *Sage Cell* do webové stránky je následující:

1. Na začátek stránky vložíme následující HTML tagy:

```
<script
 src="http://sagecell.sagemath.org/static/jquery.min.js">
</script>
<script
 src="http://sagecell.sagemath.org/
      static/embedded_sagecell.js">
</script>
<script>
  sagecell.makeSagecell({"inputLocation": ".sage"});
</script>
```

   Adresa `http://sagecell.sagemath.org` může být nahrazena adresou vlastního serveru, na kterém je dostupná spuštěná instance *Sage Cell Server*. První dva výskyty tagu `<script>...</script>` v tomto kódu zajišťují načítání skriptů v jazyce JavaScript, třetí pak slouží k převedení všech částí HTML kódu s třídou `sage` na *Sage Cell*.

   Pokud vkládáme *Sage Cell* do stránek se složitějším nastavením stylů, jako blog, či deck.js prezentace, hrozí zde konflikt mezi stylem stránky a stylem *Sage Cell*, proto je možné pod výše uvedené tagy připojit ještě čtvrtý řádek, který zajistí načtení speciálního CSS souboru pro *Sage Cell* objekty.

```
<link rel="stylesheet" type="text/css"
 href="
   http://sagecell.sagemath.org/static/sagecell_embed.css
 ">
```

2. Vložíme do stránek kód, který chceme vyhodnotit pomocí *Sage*. Kód je obalem tagem `<script>`, což zapříčiňuje, že nebude zpracováván jako HTML. Blok s třídou `sage` způsobí převedení obsahu uvnitř tagů `<div></div>` na *Sage Cell*.

```
<div class="sage">
   <script type="text/x-sage">
Sage kód k~vyhodnocení
</script>
</div>
```

Vkládání *Sage Cell* buňek do webové stránky lze dále upravovat. Je možno například ovlivnit umístění formuláře, umístění výstupu, použití konkrétního editoru zdrojového kódu, text tlačítka pro vyhodnocení kódu, implicitní kód v poli a další.

3.2.2 Grafy

Vypočítaná data je možno vynášet do grafů [9], [7], [16]. Kromě například vykreslení grafu funkce, či parametrických grafů umožňuje *Sage* vynášet do grafu i pole bodů, nebo hodnoty v matici. K těmto účelům slouží funkce `list_plot` a `matrix_plot`.

Funkce `list_plot` vytvoří graf z pole bodů. Jako data na vstupu o čekává funkce pole hodnot, nebo pole dvojic hodnot. V případě, že dostane jednorozměrné pole hodnot, použije tyto hodnoty jako souřadnice na ose y a souřadnice na ose x doplňuje automaticky zvyšováním hodnoty o konstantní krok. V případě, že dostane pole dvojic hodnot, vynáší tyto hodnoty do grafu tak, že první hodnota z dvojice představuje souřadnici na ose x a druhá hodnota z dvojice souřadnici na ose y. Funkce `matrix_plot` pracuje s dvourozměrnou maticí nebo dvourozměrným polem. V případě husté matice je každému prvku přidělena barva vzhledem k poměru jeho hodnoty vůči hodnotám ostatních prvků. V případě řídké matice barva pouze indikuje, zda-li má prvek nulovou, nebo nenulovou hodnotu.

3.3 Programování v Sage

Systém *Sage* používá pro své příkazy syntaxi jazyka Python [10] a rovněž podporuje všechny příkazy tohoto jazyka. To v podstatě znamená, že veškeré skripty napsané v jazyce Python jsou se *Sage* kompatibilní. *Sage* však má ještě svou vlastní modifikaci jazyka Python zvanou *Cython* a také vlastní příkazy.

Jazyk *Cython* je programovací jazyk založený na jazyku Python. Oproti jazyku Python obsahuje navíc některá rozšíření, zejména možnost deklarace statických datových typů, stává se tak vlastně nadmnožinou jazyka Python, což zahrnuje možnost vysokoúrovňového, objektově orientovaného, funkcionálního a dynamického programování. Zdrojové kódy jazyka *Cython* jsou převedeny na optimalizovaný kód jazyka C/C++ a posléze zkompilovány a použity jako rozšiřující moduly pro Python. Díky tomu je možno udržet vysokou programátorskou produktivitu jazyka Python a zároveň výpočetní efektivitu jazyka C/C++.

V případě, že potřebujeme používat externí funkce napsané v jazyce C, máme v zásadě dvě možnosti, jak toho docílit. V první řadě můžeme vytvořit soubor v jazyce *Cython*, v němž deklarujeme funkce, jejichž kód se nachází v externích zdrojových souborech, a v němž také tyto zdrojové soubory importujeme. Výsledný kód pak načteme příkazem `attach` v *Sage* a funkce deklarované v *Cython* skriptu tak můžeme používat i v *Sage*. V druhé řadě zdrojový kód zkompilujeme a importujeme jej jako knihovnu pro jazyk Python. Pro předávání hodnot můžeme použít knihovnu *CTypes* [5] jazyka Python.

V prvním případě je výhodné, že veškerou režii s definicemi a deklaracemi funkcí provedeme v *Cython* skriptu a ten pak k *Sage* pouze připojíme. Funkce v jazyce C navíc není potřeba kompilovat, odpadá tedy problém s kompatibilitou přeloženého kódu na jiných systémech. *Cython* navíc umožňuje načítání funkcí v jazyce C i v jazyce C++.

Pro představu mějme následující příklad vlastní funkce na výpočet druhé mocniny přirozeného čísla:

Mějme ve stejném adresáři následující dva soubory:

- `priklad.c` s obsahem:

```
int mySquare(int x) {
   return x*x;
}
```

- `priklad.spyx` s obsahem:

```
cdef extern from "priklad.c":
    int mySquare(int x)

def square(x):
         return mySquare(x)
```

Potom bude fungovat následující kód v *Sage*:

```
@attach /cesta/k/priklad.spyx
mySquare(10)
```

Výstupem bude číslo `100`. Po připojení dynamické knihovny příkazem `attach` je možné používat funkce z této knihovny kdykoliv v rámci následujícího kódu.

V druhém případě je sice rovněž možné pro přehlednost některé části kódu umístit do Python skriptu, ale přibývá nutnost přeložení souborů

s funkcemi v jazyce C jako sdílené knihovny. Zde existuje riziko nekompatibility kompilovaných kódů s některými systémy. Použití knihovny funkcí *CTypes* jazyka Python zajistí, že můžeme využívat standardní datové typy jazyka C (viz tabulka 3.1), kromě struktur, či svazků s bitovými poli tak, aby byla do funkce v *Sage* předána hodnota. Ačkoliv to může fungovat na některých 32 bitových x86 architekturách, není tato funkcionalita garantována. Tyto struktury a svazky mohou být ovšem předávány do funkce v *Sage* v podobě ukazatelů.

Jako knihovna může být použit zdrojový kód jakékoliv funkce v jazyce C, není nutno do něj vkládat speciální hlavičkové soubory, ani dodržovat jiná pravidla. Kód však nesmí obsahovat žádný prvek jazyka C++ a musí být překládán kompilátorem `gcc`. Kompilace proběhne tak, že se ze zdrojového souboru nejdříve vytvoří překladem s přepínačem `-fPIC` soubor s příponou `.o`.

```
gcc -fPIC -c mysquare.c -o mysquare.o
```

Poté se ze souboru s příponou `.o` vytvoří překladem s přeínačem `-shared` sdílený objekt – dynamickou knihovnu `.so`.

```
gcc -shared mysquare.o -o mysquare.so
```

V *Sage* poté vytvoříme objekt CDLL, přes který bude možno přistupovat k funkcím v dynamické knihovně.

```
my_square_library=ctypes.CDLL("/cesta/k/mysquare.so")
```

Přístup k funkcím v této knihovně je pak snadný, k vzorové funkci `square` se přistoupí následujícím způsobem:

```
my_square=my_square_library.square
fermi.restype=ctypes.c_int
```

Tabulka 3.1: Datové typy knihovnoy CTypes a jejich kevivalenty v jazycích C a Python

Typ z ctypes	**ekvivalent v C**	**ekvivalent v Python**
`c_bool`	`_Bool`	`bool` (1)
`c_char`	`char`	1 znakový `string`
`c_wchar`	`wchar_t`	1 znakový `unicode string`
`c_byte`	`char`	`int/long`
`c_ubyte`	`unsigned char`	`int/long`
`c_short`	`short`	`int/long`
`c_ushort`	`unsigned short`	`int/long`
`c_int`	`int`	`int/long`
`c_uint`	`unsigned int`	`int/long`
`c_long`	`long`	`int/long`
`c_ulong`	`unsigned long`	`int/long`
`c_longlong`	`__int64` nebo `long long`	`int/long`
`c_ulonglong`	`unsigned __int64` nebo `unsigned long long`	`int/long`
`c_float`	`float`	`float`
`c_double`	`double`	`float`
`c_longdouble`	`long double`	`float`
`c_char_p`	`char *` (ukončený `NULL`)	`string` or `None`
`c_wchar_p`	`wchar_t *` (ukončený `NULL`)	`unicode` nebo `None`
`c_void_p`	`void *`	`int/long` nebo `None`

Kapitola 4

Vytvoření interaktivní webové demonstrace

4.1 Wolfram Mathematica

4.1.1 Návrh

Pří návrhu bylo vycházeno z předešlé analýzy. Vyplývá z ní tedy, že zajímavou možností tvorby interaktivních webových demonstrací v CAS *Woflram Mathematica* (dále jen analýza), pro vytvoření webové prezentace je možné použít dokument ve formátu CDF. *Wofram Mathematica* umožňuje jednak převést do tohoto formátu běžný *Mathematica notebook* a jednak nabízí možnost vytvoření zvláštního dokumentu *Wolfram Demonstrations – Demonstrace*. Dále k vytvoření interaktivního rozhraní lze použít pouze příkaz `Manipulate`, který je také hlavním prvkem *Demonstrace*.

Při tvorbě interaktivní webové demonstrace má být využita implementace algoritmu pro výpočet časového vývoje SFUM (dále jen implementace). Tento algoritmus byl implementován v jazyce C/C++. Existují dvě možnosti, jak používat zdrojový kód v jazyce C/C++ uvnitř CAS *Wolfram Mathematica*. První z nich příkaz `CreateExecutable`, který přeloží dodaný kód při svém vyhodnocení. Tento příkaz ovšem vyžaduje dostupnost kompilátoru jazyka C, případně C++. Druhou možností je implementaci algoritmu přeložit na spustitelný soubor, jenž bude možné připojit k *Wolfram Mathematica* a komunikace s aplikací bude probíhat pomocí protokolu *MathLink*. Protože dostupnost kompilátoru jazyka C, resp. C++, není na webovém serveru vždy nezbytně garantována, bylo rozhodnuto použít variantu připojení přeloženého spustitelného kódu.

K vizualizaci získaných dat lze použít příkazy `ListPlot` a `ArrayPlot`.

První jmenovaný vynáší do grafu všechny souřadnice, které mu byly předány ve formě seznamu ve vstupním parametru. Jelikož jsou do grafu vynášeny všechny souřadnice a protože pro každý bod hledá příkaz jeho pozici v grafu zvlášť, stává se proces vyhodnocování příkazu `ListPlot` a generování výstupního grafického prvku pro řádově desítky tisíc bodů výpočetně náročným. Příkaz `ArrayPlot` generuje graf, jehož hodnoty jsou zobrazeny v nespojitém poli (nebo matici) čtverců. Tento příkaz dokáže také efektivně pracovat s řídkou maticí, převede-li se tedy množina dat vypočítaných implementací na matici, jejíž počet řádků a sloupců bude odpovídat rozměrům výstupního grafu, bude třeba vykreslit menší množství bodů a tím pádem proběhne toto vykreslování rychleji. Pro vytvoření interaktivní webové demonstrace tedy bude použit příkaz `ArrayPlot` spolu vytvořením kódu pro předzpracování dat předávaných tomuto příkazu.

4.1.2 Implementace

Nejdříve bylo nutné upravit zdrojový kód implementace, aby splňoval požadavky pro používání protokolu *MathLink* popsané v analýze. Výsledný soubor pak byl uložen jako `fermi.cpp`. Postupně byl zdrojový kód modifikován:

1. přidáním direktivy `#include "mathlink.h"` a zkopírováním souboru mathlink.h do adresáře, v němž bude upravená implementace uložena.

2. změnou návratového typu funkce `float**fermi(...)` na `void fermi(...)`.

3. doplněním manuální definice návratového typu pro *Wolfram Mathematica*, čehož bylo docíleno tímto voláním funkcí `MLPutFunction(stdlink, "List", listSize)` a `MLPutReal64List(stdlink, resArray[index], 2)` z `mathlink.h`.

   - Funkce `MLPutFunction(...)` předává pomocí protokolu *MathLink* funkci List s počtem argumentů listSize.
   - Funkce `MLPutReal64List(...)` předá *MathLink* spojení pole hodnot typu double o počtu 2 prvků, začínající na adrese `resArray[index]`.

     Ve *Wolfram Mathematica* je výstup funkce `MLPutReal64List(...)` přijat jako dvouprvkový seznam. Každé takto předávané pole je počítáno jako argument příkazu `List` volaného funkcí `MLPutFunction(...)`.

4. integrováním funkce `void deleteFermiRes(...)`, která uvolní dynamicky alokované dvourozměrné pole výsledků alokované ve funkce `void fermi(...)`, přímo do těla funkce `void fermi(...)`. Nebylo totiž třeba tuto funkci používat z *Wolfram Mathematica*, k uvolnění alokované paměti dochází před skončením funkce `void fermi(...)`. Funkce `void deleteFermiRes(...)` z argumentů počítá velikost dealokovaných polí, jenž je ve funkci `void fermi(...)` již vypočítána. Díky integrování funkce dealokační funkce do `void fermi(...)` tedy není třeba její opětovný výpočet.

5. přidáním funkce `int main(...)` s požadovaným obsahem.

6. zaměněním všech datových typů `float` na `double`.

Dále byla vytvořena šablona `fermi.tm`, jenž definuje rozhraní funkce fermi ve *Wolfram Mathematica* a která je nezbytná pro překlad kódu používajícího protokol *MathLink*. V šabloně bylo zadáno, že:

- k funkci `fermi` bude moci být po připojení přeloženého kódu přistupováno pomocí příkazu

```
Fermi[Real64, Real64, Integer,
      Integer, Integer, Real64, Integer]
```

- funkce v externím souboru se jmenuje `fermi` a má parametry (`double`, `double`, `int`, `int`, `int`, `double`, `int`) s názvy `v`, `t`, `timeToLive`, `granularity`, `distance`, `amplitude`, `func`.

- návratový typ funkce je `Manual`, tedy že nebo bude nastaven programátorem uvnitř externího kódu.

Poté byl vytvořen soubor `Makefile`, jenž umožňuje překlad všech souborů implementace použitím příkazu `make` z adresáře, kde je tento soubor uložen. Při překladu se nejdříve přeloží soubor `fermi.tm` příkazem `mprep` na `fermitm.o`, poté se soubor `fermi.cpp` přeloží na kód `fermi.o` a oba tyto `.o` soubory jsou posléze přeloženy s parametry `-lML32i3 -lm -lpthread -lrt -lstdc++` na výstupní spustitelný soubor `fermi`.

Jako další krok byl vytvořen běžný *Mathematica notebook* (dále jen *notebook*), jenž bude exportován do formátu CDF a z nějž bude možno vycházet při vytváření *Demonstrace*. Tento *notebook* obsahuje:

1. připojení přeloženého spustitelného programu pomocí příkazu `Install`, jenž obsahuje buďto absolutní cestu ke spustitelnému souboru `fermi`,

nebo relativní cestu k tomuto souboru vzhledem k aktuálnímu adresáři.

2. definici funkce `g[color_]` přidělující datům zadanou barvu na základě jejich souřadnic tak, že přidělí tuto barvu prvku v matici na řádku `Ceiling[#[[2]]/.2]` a sloupci `Ceiling[#[[1]]/.03]`. Příkaz `Ceiling` při tom vrací horní celou část přirozeného čísla, `#[[1]]` reprezentuje hodnotu první složky z dané dvojice v poli výsledků, `#[[2]]` hodnotu složky druhou.

3. příkaz `Manipulate`, jenž vytvoří inteaktivní rozhraní.

   Jelikož to *Wolfram Mathematica* umožňuje, jsou do grafu vynášena data pro více různých vstupních parametrů udávajících počáteční rychlost částice a okamžik poslední kolize částice s nepohyblivou stěnou. Přidat novou dvojici počátečních hodnot lze kliknutím do grafu přesně v bodě, jehož souřadnice mají být přidány do seznamu počátečních rychlostí a okamžiků poslední kolize částice s nepohyblivou stěnou.

   Ovládacím prvky zadanými v příkazu `Manipulate` jsou:

   - `{{r,1000,"Count of results"},1,10000,1}` – posuvník s hodnotami celých čísel od 1 do 10000 s krokem velikosti 1 a hodnotou po vytvoření inicializovanou na hodnotu 1000. Parametr `r` zadává počet kolizí částice s nepohyblivou stěnou, pro něž jsou data generována.
   - `{{gr,1,"Granularity of returned results"},1,100,1}` – posuvník s hodnotami celých čísel od 1 do 100 s krokem velikosti 1 a hodnotou po vytvoření inicializovanou na hodnotu 1. Parametr `gr` udává granularitu výsledků, které jsou odesílány na výstup výpočetní funkce a předány do *Wolfram Mathematica*.
   - `{{d,1,"Distance"},1,10000,1}` – posuvník s hodnotami celých čísel od 1 do 10000 s krokem velikosti 1 a hodnotou po vytvoření inicializovanou na hodnotu 1. Parametr `d` udává minimální vzdálenost pohyblivé stěny od nepohyblivé.
   - `{{a,1.0,"Amplitude of oscillation"},0.1,10.0}` – posuvník s hodnotami přirozených čísel od $0,1$ do $10,0$ s hodnotou po vytvoření inicializovanou na hodnotu $1,0$. Parametr a udává absolutní hodnotu amplitudy periodické funkce obsažené v předpisu pohybové funkce pohyblivé stěny.

- `{{f,0,"Oscilation function"},{0→"sinus", 1→"saw"}}` – tlačítka (přepínač), jimiž se zvolí příznak určující, zda předpis pohybové funkce pohyblivé stěny obsahuje funkcu sinus, nebo „pilovou“ funkci.
- `{{vmax,1000,"Max speed"},300,1000}` – posuvník s hodnotami celých čísel od 300 do 1000 s hodnotou po vytvoření inicializovanou na hodnotu 1000. Parametr `vmax\verb` udává maximální rychlost částice, jenž bude ještě zasnesena do grafu.
- `Button["Delete",If[Length[pt] > 1,pt = Most[pt]]]` – tlačítko, jenž odstraní poslední dvojici souřadnic z parametru `pt`, až na tu inicializační, která byla vytvořena při vyhodnocování příkazu `Manipulate`.
- `{{pt,{{0,10}}},Locator,LocatorAutoCreate→All}` – polohový bod v grafu, jehož souřadnice jsou umístěny v parametru `pt`. Pokud je změněna poloha bodu v grafu, jsou aktualizovány souřadnice v parametru `pt`. První souřadnice parametru `pt` je použita jeko vstupní parametr `t` a druhá souřadnice je použita jako parametr `v` funkce `Fermi[...]`. Při kliknutí do grafu je vytvořen další nový takový bod a jsou vygenerována data pro graf s počátkem v tomto bodě.

Pro ovládací prvky zadané funkci `Manipulate` byla ještě zakázána plynulost akce (vyhodnocování výrazu proběhne až po puštění stisknutého tlačítka myši) volbou `ContinuousAction→False`, dále bylo zajištěno nastavení první hodnoty v poli barev `col` na *černou* volbou: `Initialization:>{col = {Black}}`,
čímž je řečeno, že prví vytvořený graf bude *černé* barvy.

Výraz vyhodnocovaný příkazem `Manipulate` je tvořen následujícím kódem:

1) Ověření, zda je v seznamu barev `col` právě tolik položek, kolik datových sad (každé volání funkce `Fermi[...]` během jednoho vyhodnocení výrazu v příkazu `Manipulate` vytvoří jednu datovou sadu) je vytvořeno. Pokud je těchto sad méně, než kolik datových sad, je vygenerována nová barva ze škály RGB a uložena do seznamu barev `col`. Pokud je sad více, je poslední barva ze seznamu barev `col` odstraněna.
2) Výpočet datových sad pro každou dvojici z parametru `pt`. Každá sada dat tvoří jeden řádek tabulky uložené v proměnné `data`.
3) Určení maxima z vypočítaných dat.

4) Vygenerování grafu příkazem `ArrayPlot` s přepínači:
   - `AspectRatio→1` – poměr stran grafu bude roven 1
   - `ImageSize→{500,500}` – velikost generovaného grafu bude 500×500 pixelů.
   - `DataReversed→True` – počátek souřadnic bude v levém dolním rohu grafu (implicitně je v levém horním rohu).
   - `Frame→True` – bude zobrazen rámeček okolo grafu.

Jelikož příkaz `ArrayPlot` pracuje s dvourozměrným polem, nebo maticí, je třeba vypočítaná data předzpracovat a převést do požadovaného formátu:

1) Je vytvořena nová tabulku dat, v jejíž každém řádku je jedna sada vypočítaných dat rozšířená o složku přidělené barvy. Příkazem `Map` je aplikována funkce `g` na všechny dvojice bodů v zadané sadě vypočítaných dat, čímž je bodům přidělena barvu podle množina dat, z níž pocházejí.
2) ze získané tabulky dat je vytvořeno dvourozměrné pole příkazem `Flatten`.
3) Příkazem `DeleteDuplicates` jsou z pole odstraněny duplicitní záznamy.
4) Výsledné pole je přeměněno na řídké pole (řídkou matici) příkazem `SparseArray`.

Protože při zobrazování matice příkazem `ArrayPlot` odpovídají hodnoty na osách číslům řádků a sloupců matice, je nutné pomocí volby `FrameTicks` vytvořit novou tabulku hodnot, jenž mají být na osách zobrazovány. Tyto hodnoty, z nichž jsou číselné osy generovány, jednak odpovídají hodnotám používaným ve funkci `g` k převodu hodnot rychlosti a času na souřadnice matice a jedna jednak jsou voleny podle parametru `f`. Hustota čísel na souřadnicových osách byla doplněna na základě testování různých možností a vizuální podoby grafu při jejich použití. Náhled interaktivního rozhraní viz obrázky 4.1 a 4.2.

4.2 Sage

4.2.1 Návrh

Pří návrhu bylo vycházeno z analýzy možností tvorby imteraktivních webových rozhraní v Sage. CAS *Sage* disponuje webovým uživatelským rozhraním, rovněž existuje možnost vložit výpočetní buňky *Sage Cell*, které

jsou totožné s výpočetními buňkami webového rozhraní *Sage*, do libovolné webové stránky. Tím je umožněno prezentovat na webu libovolný *Sage* kód bez nutnosti jakýchkoliv jeho změn. Interaktivitu pak zajistí příkaz `interact` fungující ve webovém rozhraní *Sage*.

Pro vynášení vypočítaných dat vypočítaných implementací do grafu může být v *Sage* použita buďto funkce `list_plot`, nebo funkce `matrix_plot`. Tato funkce je určena pro grafické znázornění obsahu matic, a to tak, že každému číslu v matici je při vykreslování přiřazena barva z dané škály na základě hodnoty tohoto čísla. Pro reprezentaci čísla v grafu je touto barvou vyplněn čtverec na příslušných souřadnicích grafu. Čísla řádků a sloupců pak tvoří souřadnicový systém grafu, jenž takto vzniká. Funkce `matrix_plot` funguje obdobně jako příkaz `ArrayPlot` v software *Wolfram Mathematica*, tudíž byla zvolena pro vizualizaci vypočítaných dat rovněž z důvodu znázornění rozdílů mezi oběma systémy.

Jak vyplývá z dokumentace funkce `matrix_plot`, pokud je jí předána řídká matice, není možné přemístit počátek souřadnicového systému z levého horního rohu do levého dolního rohu, takto vytvořený graf by tedy byl vůči požadovanému výsledku zobrazen zrcadlově podle horizontální osy. Aby bylo docíleno podobného vzhledu obou implementací, bude tedy nutno vyhnout se předzpracování dat do řídké matice.

Sage umožňuje dva způsoby, jak použít externí kód v jazyce C. Jedním je vytvoření skriptu v jazyce *Cython*, v němž budou deklarovány funkce implementace, k nimž chceme ze *Sage* přistupovat. Jelikož však v dokumentaci *Cython*, ani *Sage*, ani v některém z kódů zveřejněných na webových stránkách *Sage Interact* komunity není popsán, ani implementován způsob, jak deklarovat funkci, jejíž návratovou hodnotou je reference na dynamicky alokované pole, nejeví se použít jazyka *Cython* jako optimální. Druhým způsobem, jak pracovat s externím zdrojovým kódem, je přeložit tento kód a vytvořit dynamickou knihovu, dále importovat v *Sage* kódu knihovnu `ctypes` a jejím prostřednictvím pracovat s daty vyprodukovanými knihovními funkcemi. Používání `ctypes` má rovněž svá omezení stran kompatibility s objekty jazyka *Python*, ovšem pro práci s hodnotou předávanou ve formátu datového typu jazyka C se jeví jako optimální.

4.2.2 Implementace

Zdrojový kód implementace algoritmu nebylo třeba měnit, nebo doplňovat o zvláštní struktury, které by si vyžádalo použití knihovny *CTypes*, nebo propojení se *Sage*. Jedinou nutností bylo je přeložení kódu kompilátorem `gcc`, nikoliv `g++`.

Prvním krokem vytváření interaktivní webové demonstrace bylo přelo-

žení implementace algoritmu v souboru `fermi.c` na dynamickou knihovnou `fermi.so`. Nejdříve se soubor přeložil kompilátorem `gcc` s přepínači `-fPIC -o` na `fermi.o`. Tento výstupní soubor byl pak kompilátorem `gcc` s přepínačem `-shared -o` přeložen na požadovanou dynamickou knihovnu `fermi.so`.

Následně byl vytvořen *Sage* kód s příkazem `interact`. Ještě před samotným vytvořením byl definován datový typ pro referenci na dvourozměrné dynamicky alokované pole hodnot typu `c_float`, jenž reprezentují datový typ `float` z jazyka C, jako

```
float_pointer_pointer=POINTER(POINTER(c_float))
```

a pomocí `ctypes.CDLL` načtena dynamická knihovna `fermi.so` do proměnné `my_library`. Do proměnné `fermi` byl přiřazen odkaz na funkci z dynamické knihovny
`my_library.fermi`, dále bylo vytvořeno rozhraní pro přístup k hlavní výpočetní funkci implementace algoritmu, a to pomocí přiřazení
`fermi.restype=float_pointer_pointer`, čímž byl vlastně definován návratový typ této funkce jako ukazatel na dvourozměrné dynamicky alokované pole z dynamické knihovny jazyka C. Poté bylo ještě třeba vytvořit přístup k funkci,jenž má za úkol uvolňovat alokovanou paměť v implementaci algoritmu z knihovny, to bylo provedeno přiřazením `fermi.so free_data=my_library.deleteFermiRes`.

Za příkazem `@interact` (`@` je používáno u některých příkazů ve webovém rozhraní *Sage*) byla definována funkce pro vytvoření interaktivní demonstrace `fermi_acceleration` s parametry:

- `v = slider(srange(0.1,1000.0,0.1), default = 10.0, label="v0")` – posuvník, jímž se nastavuje rychlost na začátku výpočtu.
- `t = slider(srange(0.0,6.28,0.1), default = 0.0, label="t0")` – posuvní, jímž se nastavuje váze periody v okamžiku začátku výpočtu.
- `r = slider([100..10000], default = 1000, label="# of results")` – posuvník, jímž se nastavuje počet odrazů od nepohyblivé stěny, pro které mají být vypočítána data.

- `gran = slider([1..100], default = 1, label="granuality of results")` – posuvník, jímž se nastavuje granularita dat předaných implementací algoritmu pro výpočet dat.
- `d = slider([1..100], default = 1, label="distance")` – posuvník, kterým se nastaví minimální vzdálenost, v níž se může nacházet pohyblivá stěna od nepohyblivé.
- `a = slider(srange(0.1,10.0,0.1), default = 1.0, label="amplitude")` – posuvník, jímž se nastaví amplituda kmitu periodické funkce obsažené v předpisu pohybové funkce pohyblivé stěny.
- `o = selector(['sinus','saw'], buttons=True, label="function")` – dvojice tlačítek, jimiž se zvolí periodická funkce obsaženáv předpisu pohybové funkce pohyblivé stěny. Saw = „pilová funkce“, Sin = sinus.
- `vmax = slider(srange(50.0,10000.0,10.0), default = 50.0, label="max velocity")` – posuvník, jímž se nastavuje maximální rychlost částice, jenž je vynášena do grafu.

V této funkci jsou vypočítávána data voláním funkce `fermi` s danými parametry vytvořenými konstruktory objektů třídy `ctypes`. Výsledek funkce `fermi` je uložen v proměnné `data`. Dále jsou nastaveny rozsahy dat pro graf – délka periody je uložena v proměnné `xrange` a maximální rozsah a zároveň maximální rychlost vynášená do grafu v proměnné `yrange`. Poté jsou nastaveny rozměry matice `row` (počet řádků) a `col` (počet sloupců) na čtvercovou matici o 250 sloupcích i řádcích a vypočítána šířku (`xdelta`) a výšku (`ydelta`) prvku matice vzhledem k zadanému počtu řádků a známému rozsahu hodnot x a y. Nakonec je vytvořena matice přirozených čísel o rozměrech `row` řádků a `col` sloupců. Tato matice není v konstruktoru nastavena jako řídká matice.

Poté je třeba naplnit matici daty. V cyklu pro každý prvek z proměnné data je na danou pozici v matici přiřazena hodnota 1. Správná pozice hodnoty v matici se vypočítá podílem první složky položky výsledných dat

s proměnnou `yrange`, čímž je určen řádek matice, a podílem druhé složky položky vypočtených dat s proměnnou `xrange`, čímž je je určen sloupec matice.

Dále je generován graf z připravené matice příkaze `matrix_plot` s parametry:

- `m` – proměnná `m` obsahuje předpřipravenou matici s daty
- `origin='lower'` – parametr `origin` říká, zda-li je počátek souřadnic v levém horním (`upper`) nebo dolním (`lower`) rohu.
- `cmap=get_cmap(['white','red'])` – definuje barevnou paletu použitnou k vykreslení bodu. Pro tuto potřebu byla definována nová paleta skládající se z *bílé* (odpovídá hodnotě 0) a *červené* (odpovídá hodnotě 1). Pozadí grafu (pozice v matici s hodnotou 0) tedy zůstane *bílé* a vypočítané body vynášené do grafu budou vykresleny *červeně* (pozice v matici s hodnotou 1).
- `axes=False`, – Vypne zobrazování číselných os. Jelikož nebyl nalezen způsob, jak ovlivnit rozsahy čísel [8] na osách ve funkci `matrix_plot`, aniž by se měnil výstupní graf, bylo rozhodnuto, že, než zobrazovat chybné údaje, je lepší nezobrazovat žádné
- `frame=False` – Vypne zobrazování rámečku. Viz předchozí parametr.

Výstpní funkce `matrix_plot` je uložen do proměnné `dem`. Obsah proměnné `dem` je následně zobrazen funkcí `show`, jejímž parametrem je právě proměnná `dem`. Po skončení výpočtu je uvolněna alokovaná paměť s výsledky příkazem:

```
free_data(data, c_int(r), c_int(gran))
```

Nakonec byla vytvořena jednoduchá webová stránka s vestavěnou výpočetní buňkou *Sage Cell* s kódem interaktivní webové demonstrace *Fermiho akcelerace*. Kód demonstrace napsaný v *Sage* byl umístěn mezi tagy `<div>` a `</div>` třídy `sage` jako skript. Objevil se ovšem problém při jeho vyhodnocení, protože funkce `ctypes.CDLL` nedokáže připojit soubor z online umístění. Náhled interaktivního rozhraní viz obrázky 4.3 a 4.4.

4.3 Porovnání implementací a vyhodnocení

Formát CDF deleguje webové aplikaci takřka všechny výpočetní schopnosti softwaru *Wolfram Mathematica*, k fungování této aplikace je však nutné, aby byl v počítači uživatele nainstalován *Wolfram CDF Player* – zdarma distribuovaný program, resp. zásuvný modul do webového prohlížeče. Formát

CDF umožňuje vytvořit aplikaci pro *Wolfram Demonstrations Project*, která může být zveřejněna v katalogu na webu projektu, ale rovněž je možno do CDF formátu exportovat obyčejný *Mathematica notebook* a vytvořit z něj modul, jenž je možné vložit do libovolných webových stránek.

Sage je vybaven součástí *Sage Cell Server*, která umožňuje pomocí vložení několika skriptů v jazyce JavaScript vložit *Sage* kód přímo do webové stránky, pouze ohraničený tagy `<div></div>` se speciální třídou, podle níž vykonávaný JavaScriptový program pozná, že se jedná o text, jenž má být vyhodnocen v *Sage* a nahradí tento text buňkou *Sage Cell* s původním kódem uvnitř. Takto vytvořený objekt navíc nepotřebuje žádná speciální doplňky, ani moduly na straně klienta – návštěvníka webu. Jediné, co musí návštěvník webu udělat, je kliknout na tlačítko pro vyhodnocení kódu.

Interaktivní rozhraní lze ve *Wolfram Mathematica* vytvořit pomocí příkazu `Manipulate`, která je zároveň základním stavebním kamenem pro *Demonstrace*. Výsledný objekt vytvořený touto funkcí působí celistvě a připomíná spíše applet, či speciální modul. Za pomoci formátu CDF jej lze navíc používat pomocí *Wolfram CDF Playeru* a vložit do webových stránek.

Pro vytváření interaktivních prvků v *Sage* je použit příkaz interact. Použitím tohoto příkazu před definicí funkce autor kódu zajistí, že jsou-li v kódu funkce použity nějaké interaktivní prvky, jako posuvníky, tlačítka, textová pole, rozbalovací menu atd., funkce je vyhodnocena pokaždé, když některý z těchto prvků zaznamená interakci s uživatelem.

Pokud je ve *Wolfram Mathematica* v rámci příkazu `Manipulate` použit ovládací prvek `Locator`, je software schopen přečíst souřadnice uvnitř výsledného objektu příkazu `Manipulate`, na kterých se objekt `Locator` nachází. Pokud se souřadnice tohoto objektu změní, program to rozezná a výraz v příkazu `Manipulate` opět vyhodnotí, ovšem s aktuálními daty z ovládacích prvků, stejně, jako když se například posune jezdcem posuvníku, či klikne na tlačítko. Toto je umožněno díky skotečnosti, že grafy a grafické prvky jako takové jsou ve *Wolfram Mathematica* reprezentovány jako speciální objekty, nikoliv například jako bitmapy, a jejich náhledy jsou dynamicky generovány.

V *Sage Interact* není možné použít ovládací prvek, jako je objekt `Locator` programu *Wolfram Mathematica*. Takový prvek není implementován, neboť grafy jsou generované jako bitmapy a nesleduje je žádný skript nebo program, který by zajistil snímání pozice kurzoru myši v okamžiku, když je stisknuto tlačítko. *Sage* tedy neumožňuje měnit parametry jinak, než tlačítky, posuvníky, textovými poli atd.

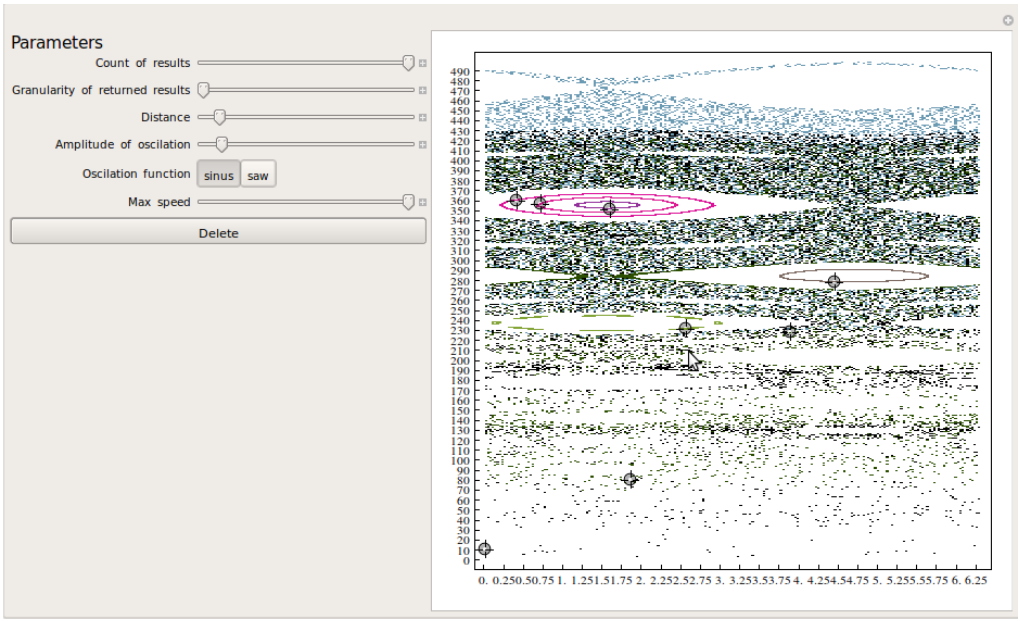
Wolfram Mathematica přímo nenabízí žádný speciální efektivně vyhodnocovaný příkaz nebo funkci k rychlé vizualizaci velkého množství získaných dat, jakým jsou právě například data pro znázornění časového vývoje

SFUM. Pro zobrazení grafu sestaveného s pole hodnot, či souřadnic, nabízí příkaz `ListPlot`. Větší rychlosti při vykreslování dat však lze dosáhnout, pokud se tato data nejdříve předzpracují a poté předají jako matici, či dvourozměrné pole příkazu `ArrayPlot`.

Ani systém *Sage* přímo nenabízí žádné sofistikované funkce pro vykreslení velkého množství dat do grafu. K vykreslení grafu z pole zadaných hodnot, či pole zadaných souřadnic používá funkci `list_plot`, která však požaduje, datový objekt, jenž je jí předáván, implementoval funkci `len()`, vracející délku pole. V případě, že jdou data předzpracována, je zde stejně jako u *Wolfram Mathematica* možnost předat předzpracovaná data v podobě matice funkci `matrix_plot`. Tato funkce však, na rozdíl od příkazu `ArrayPlot` ve *Wolfram Mathematica*, neumožňuje ovlivnit hodnoty čísel zobrazovaných na souřadnicových osách. Číslování os tedy přímo odpovídá číslování řádků a sloupců vykreslované matice. Stejně jako v případě `ArrayPlot` je u grafu generovaného funkcí `matrix_plot` počátek souřadnic umístěn v levém horním rohu. Změnit umístění počátku souřadnic lze jak v případě příkazu `ArrayPlot`, tak i v případě funkce `matrix_plot`, ovšem funkce `matrix_plot` neumožňuje měnit pozici počátku souřadnic, pokud pracuje s řídkou maticí. Jelikož je však *Sage* distribován jako Open Source a jelikož podporuje popužívání skriptů napsaných v jazyce Python, lze pro potřeby vykreslování velkého množství získaných dat do grafu vytvořit skript, nebo upravit některý stávající, kupříkladu funkci `list_plot`.

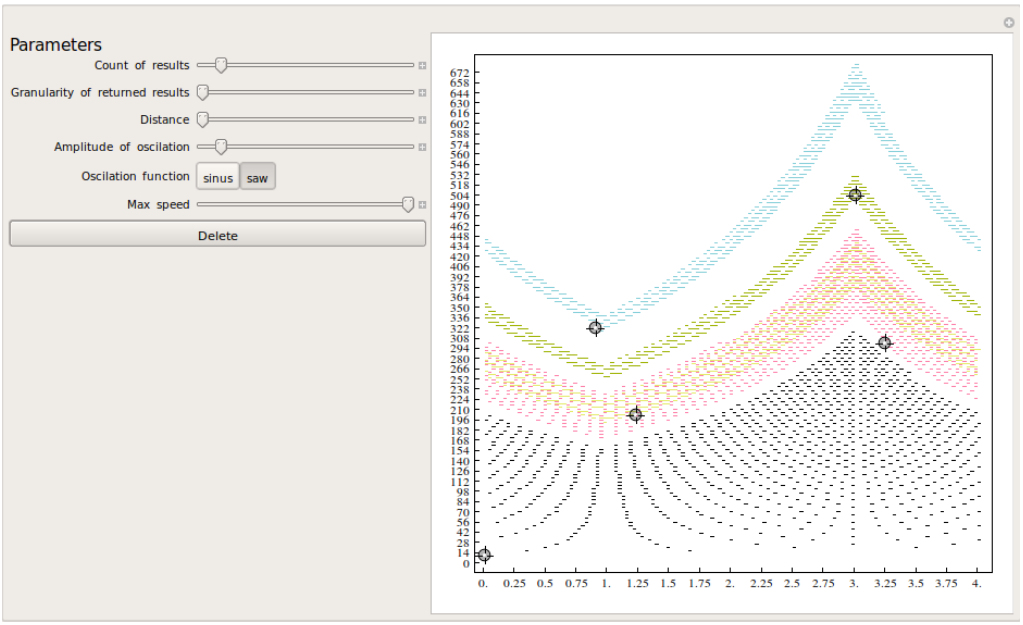
CAS *Wolfram Mathematica* umožňuje programování pomocí svých vestavěných funkcí, což může být užitečné pro některé jednodušší algoritmy s malou výpočetní složitostí, neboť tyto algoritmy jsou kvůli nutnosti pracovat s *Mathematica* objekty výpočetně i paměťově náročnější, než kdyby byly naprogramovány v nízkoúrovňovém jazyce, jako například jazyk C. Právě proto se pro algoritmy s větší výpočetní a paměťovou složitostí vyplatí – a *Wolfram Mathematica* to umožňuje – napsat je v jazyce C, načítat jej do *Mathematica notebooku* a buďto tento kód při vyhodnocování buňky zkompilovat, nebo importovat již zkompilovaný modul komunikující pomocí protokolu *MathLink*.

Sage rovněž nabízí možnost připojení externích zdrojových kódů, a to jednak pomocí skriptů v jazyce *Cython* a jednak pomocí knihovny *CTypes* a překladu externího kódu na dynamickou knihovnu. Výhodami jazyka *Cython* jsou především efektivnost programování a přenositelnost kódu díky tomu, že se jedná o rozšířeno verzi Pythonu a také, díky skutečnosti, že kód v jazyce *Cython* je před spuštěním převeden do jazyka C/C++ a přeložen, se zvýší efektivita vykonávání programu a přiblíží se k efektivitě ekvivalentu napsaného v nízkoúrovňovém jazyce. Knihovna *CTypes* na druhou stranu za umožňuje pracovat v jazyce Python s datovými typy jazyka C, tak, jak



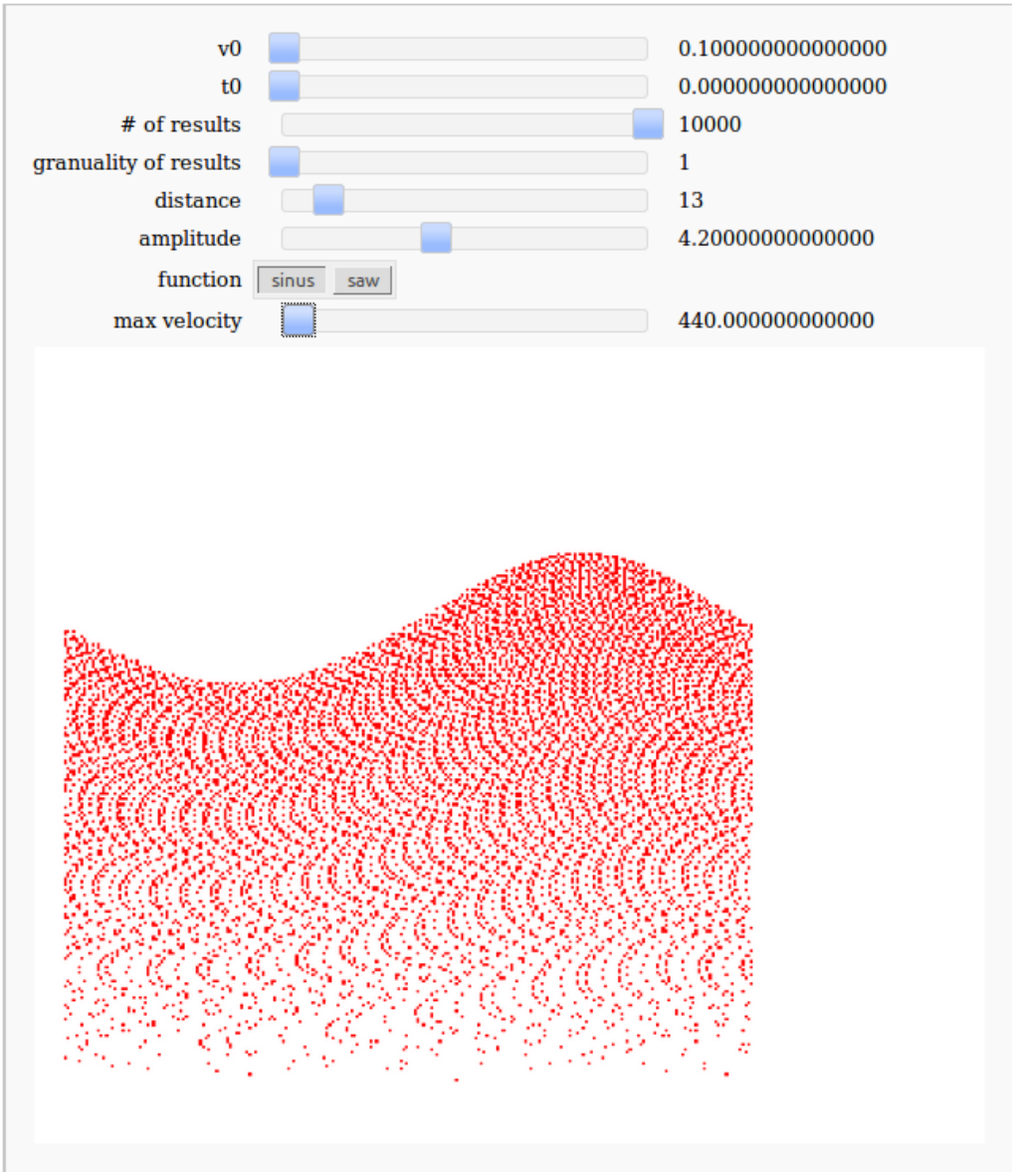
Obrázek 4.1: Ukázka interaktivního rozhraní ve Wolfram Mathematica s funkcí sinus.

je externí program předává, neumí ovšem pracovat s jazykem C++.



Obrázek 4.2: Ukázka interaktivního rozhraní ve Wolfram Mathematica s „pilovou“ funkcí.

Obrázek 4.3: Ukázka interaktivního rozhraní v Sage s „pilovou“ funkcí.



Obrázek 4.4: Ukázka interaktivního rozhraní v Sage s funkcí sinus.

Závěr

Cílem této práce bylo zaprvé navrhnout a implementovat algoritmus pro výpočet časového vývoje modelu *Fermiho akcelerace*. Zadruhé vytvořit interaktivní webovou demonstraci *Fermiho akcelerace* prostřednictvím CAS *Wolfram Mathematica* a *Sage*. A zatřetí analyzovat možnosti tvorby interaktivních webových demonstrací prostřednictvím výše zmíněných dvou CAS.

V první části práce byl autorem nastudován model SFUM a jeho vlastnosti. Poté byl navržen algoritmus na generování dat časového vývoje SFUM upravený pro účely práce tak, že jeho asymptotická složitost je nejhůře lineární pro různé počty zadávaných kolizí částice s nepohyblivou stěnou. Navržený algoritmus byl posléze implementován v jazyce C. Tento cíl byl tedy splněn, ačkoli by se dala implementace algoritmu ještě rozšířit například o možnost obecněji definovat příslušnou periodickou funkci.

Analýza obou CAS ve druhé a třetí části práce by se dala zhodnotit jako úspěšná. Bylo zjištěno, že výhodami *Sage* je jeho dostupnost, neboť se jedná o Open Source, a také skutečnost, že pro vytváření interaktivního webového obsahu není třeba, aby měl návštěvník webu v počítači nainstalován speciální software. Pro některé autory kódů pak může být výhodou i kompatibilita jazyka Python s kódem Sage (ale ne naopak). Na druhé straně jsou jeho nevýhodami poměrně chudá nabídka ukázek a příkladů a méně možností interakce s rozhraním vytvořeným příkazem `interact`. Výhodou *Wolfram Mathematica* potom je široká škála možností příkazu `Manupulate` pro tvorbu interaktivních rozhraní. Nevýhodou, a to docela zásadní, je pak skutečnost, že aby mohla být interaktivní webová demonstrace na straně klienta zobrazena, musí být v jeho počítači nainstalován speciální, zdarma distribuovaný software.

V závěrečné části se pak práce věnovala vytvoření interaktivních webových demonstrací *Fermiho akcelerace* v *Sage* a *Wolfram Mathematica*. Tento cíl byl naplněn jen částečně, neboť demonstrace sice základní funkci, tj. vygenerovat a vykreslit data na základě interaktivně zadávaných parametrů, zvládají, ale mohly by doznat mnoha vylepšní a dopracování.

Rozhraní ve *Wolfram Mathematica* sice funguje, ale výraz uvnitř funkce `Manupulate` by mohl doznat ještě mnoha změt k zefektivnění výpočtu, zejména pak jsou-li přidávány a odebírány sady zobrazovaných dat. Také vizuální stránka uživatelského rozhraní a jeho popis jsou spíše provizorní.

Interaktivní webová demonstrace v *Sage* také není v optimálním stavu. Mimi množství dalších funkcionalit by například ještě bylo vhodné přidat možnost vyhledat podle hodnot pro vytvoření grafu zajímavé počáteční rychlosti a časy, resp. prázdná místa v grafu. Uživatelské rozhraní v *Sage* je rovněž jistým provizoriem.

Práce na obou rozhraních však pomohla upozornit na vzájemné rozdíly mezi jednotlivými CAS a také na jejich individuální vlastnosti, ať už kladné, či záporné. Práce na interaktivních webovách demonstracích by tedy dále mohla pokračovat směrem k vytvoření *Demonstrace* a její publikování na webu *Wolfram Demonstrations Project* a vytvoření interaktivní demonstrace v *Sage*, která by mohla být publikována na webu komunity *Sage Interact*.

Literatura

[1] FERMI, E.: On the Origin of the Cosmic Radiation. *Physical Review*, ročník 75, č. 8, 1949: s. 1169 – 1174, ISSN 0031-899x, doi:10.1103/PhysRev.75.1169. Dostupné z: http://link.aps.org/doi/10.1103/PhysRev.75.1169

[2] LIEBERMAN, M. a. A. L.: Stochastic and Adiabatic Behavior of Particles Accelerated by Periodic Forces. *Physical Review A*, ročník 5, č. 4, 1972: s. 1852 – 1866, ISSN 0556-2791, doi:10.1103/PhysRevA.5.1852. Dostupné z: http://link.aps.org/doi/10.1103/PhysRevA.5.1852

[3] NAPLEKOV, A. V. T. a. V. V. Y., D. M.: Minimal model of the Fermi acceleration. *Technical Physics*, ročník 55, č. 5, 2010: s. 601 – 612, ISSN 1063-7842, doi:10.1134/S1063784210050038. Dostupné z: http://www.springerlink.com/index/10.1134/S1063784210050038

[4] PRESS, W. T. V. a. B. P. F., William H.: *Numerical recipes in C the art of scientific computing*, kapitola Root Finding and Nonlinear Sets of Equations. Cambridge: University Press, 1996, ISBN 0-521-43108-5, s. 347 – 383.

[5] Python Software Foundation: *15.17. ctypes — A foreign function library for Python*. 2013, [cit. 2013-05-17]. Dostupné z: http://docs.python.org/2/library/ctypes.html

[6] Robert Bradshaw, Stefan Behnel, et al.: *Cython's Documentation*. 2013, [cit. 2013-05-17]. Dostupné z: http://docs.cython.org/

[7] The Sage Group, plc.: *2D Plotting*. 2013, [cit. 2013-05-17]. Dostupné z: http://www.sagemath.org/doc/reference/plotting/sage/plot/plot.html

[8] The Sage Group, plc.: *Graphics objects*. 2013, [cit. 2013-05-17]. Dostupné z: http://www.sagemath.org/doc/reference/plotting/sage/plot/graphics.html

[9] The Sage Group, plc.: *Plotting*. 2013, [cit. 2013-05-17]. Dostupné z: http://www.sagemath.org/doc/tutorial/tour_plotting.html

[10] The Sage Group, plc.: *Programming*. 2013, [cit. 2013-05-17]. Dostupné z: http://www.sagemath.org/doc/tutorial/programming.html

[11] The Sage Group, plc.: *Sage Cell Server*. 2013, [cit. 2013-05-17]. Dostupné z: http://www.sagemath.org/eval.html

[12] The Sage Group, plc.: *Sage Feature Tour*. 2013, [cit. 2013-05-17]. Dostupné z: http://www.sagemath.org/tour.html

[13] The Sage Group, plc.: *Sage Interactions*. 2013, [cit. 2013-05-17]. Dostupné z: http://wiki.sagemath.org/interact/

[14] The Sage Group, plc.: *Sage Reference v5.9*. 2013, [cit. 2013-05-17]. Dostupné z: http://www.sagemath.org/doc/reference/index.html

[15] The Sage Group, plc.: *sagemath/sagecell . GitHub*. 2013, [cit. 2013-05-17]. Dostupné z: https://github.com/sagemath/sagecell

[16] The Sage Group, plc.: *Tutorial for Advanced 2d Plotting*. 2013, [cit. 2013-05-17]. Dostupné z: http://www.sagemath.org/doc/prep/Advanced-2DPlotting.html

[17] ULAM, S. M.: On Some Statistical Pproperties of Dynamical Systems. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability: Svazek 1*, Berkeley: University of California Press, 1961, s. 315 – 320. Dostupné z: http://www.google.cz/books?id=OZF2WtUKB44C&lpg=PA315&ots=5nNk9v2cah&dq=ON%20SOME%20STATISTICAL%20PROPERTIES%20OF%20DYNAMICAL%20SYSTEMS%20S.%20M.%20%20Ulam&lr&hl=cs&pg=PA315#v=onepage&q&f=false

[18] WAGNER, D. B.: Power Programming: MathLink Mode. *The Mathematica Journal*, ročník 6, č. 3, 1996: s. 44 – 57. Dostupné z: http://www.mathematica-journal.com/issue/v6i3/columns/wagner/contents/63wagner.pdf

[19] Wolfram Research, Inc.: *ArrayPlot*. 2013, [cit. 2013-05-17]. Dostupné z: http://reference.wolfram.com/mathematica/ref/ArrayPlot.html

[20] Wolfram Research, Inc.: *CDF (.cdf)*. 2013, [cit. 2013-05-17]. Dostupné z: http://reference.wolfram.com/mathematica/ref/format/CDF.html

[21] Wolfram Research, Inc.: *CreateExecutable*. 2013, [cit. 2013-05-17]. Dostupné z: http://reference.wolfram.com/mathematica/ref/CreateExecutable.html

[22] Wolfram Research, Inc.: *Document Formats*. 2013, [cit. 2013-05-17]. Dostupné z: http://reference.wolfram.com/mathematica/guide/DocumentFormats.html

[23] Wolfram Research, Inc.: *Install*. 2013, [cit. 2013-05-17]. Dostupné z: http://reference.wolfram.com/mathematica/ref/Install.html

[24] Wolfram Research, Inc.: *Installing Existing MathLink-Compatible Programs*. 2013, [cit. 2013-05-17]. Dostupné z: http://reference.wolfram.com/mathematica/tutorial/InstallingExistingMathLinkCompatiblePrograms.html

[25] Wolfram Research, Inc.: *Introduction to MathLink*. 2013, [cit. 2013-05-17]. Dostupné z: http://reference.wolfram.com/mathematica/tutorial/IntroductionToMathLink.html

[26] Wolfram Research, Inc.: *ListPlot*. 2013, [cit. 2013-05-17]. Dostupné z: http://reference.wolfram.com/mathematica/ref/ListPlot.html

[27] Wolfram Research, Inc.: *Manipulate*. 2013, [cit. 2013-05-17]. Dostupné z: http://reference.wolfram.com/mathematica/ref/Manipulate.html

[28] Wolfram Research, Inc.: *MathLink API*. 2013, [cit. 2013-05-17]. Dostupné z: http://reference.wolfram.com/mathematica/guide/MathLinkAPI.html

[29] Wolfram Research, Inc.: *Uses and Examples of the Computable Document Format (CDF)*. 2013, [cit. 2013-05-17]. Dostupné z: http://www.wolfram.com/cdf/uses-examples/

Příloha **A**

Seznam použitých zkratek

CAS Computer algebra system

CDF Computable Document Format

CSS Cascading style sheets

FUM Fermi-Ulam model

SFUM Simple Fermi-Ulam model

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `implemention` ..... soubory a adresáře s implementacemi
  - `mathematica` ..... adresář implementace v Mathematica
    - `example` ..... příklad interaktivní webové prezentace
    - `fermi` ..... interaktivní demonstrace Fermiho akcelerace v Mathematica
  - `sage` ..... adresář implementace v Sage
    - `example` ..... příklad interaktivní webové prezentace
    - `fermi` ..... interaktivní demonstrace Fermiho akcelerace v Sage
  - `fermi.c` ..... zdrojový kód výpočetního programu v jazyve C
- `src` ..... soubory k vygenerování práce
  - `BP_Kotýnek_Vladimír_2013.tex` ..... šablona ve formátu LaTeX
  - `*` ..... ostatní soubory potřebné k šabloně
- `text` ..... text práce
  - `BP_Kotýnek_Vladimír_2013.pdf` ..... text práce ve formátu PDF