Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

MREŽE PROSTE METODE NA VZPOREDNIH RAČUNALNIKIH

doktorska disertacija

Marjan Šterk

mentor: prof. dr. Borut Robič, somentor: doc. dr. Roman Trobec

Ljubljana, 2005

Izjava

Doktorska disertacija z naslovom "Mreže proste metode na vzporednih računalnikih" je v celoti rezultat mojega dela pod mentorstvom prof. dr. Boruta Robiča in doc. dr. Romana Trobca. Vsi dosežki, na katerih sem gradil, so dosledno označeni z navedbo virov literature.

Marjan Šterk

Zahvala

Za mentorstvo se zahvaljujem prof. dr. Borutu Robiču in doc. dr. Romanu Trobcu.

Delo ne bi nastalo brez prof. dr. Božidarja Šarlerja, ki mi je predstavil mreže proste metode, in bi bilo manj dovršeno brez komisije za oceno disertacije in njenih številnih konstruktivnih pripomb. Tudi sodelavcem na odseku E6 na Institutu Jožef Stefan, še posebej dr. Romanu Trobcu, Igorju Rozmanu in Sreču Plevelu, gre zahvala za podporo pri delu in zanimive razprave v preteklih letih. Nenazadnje naj omenim starše in Majo ter njihovo vzpodbudo in potrpljenje v času nastajanja dela.

Raziskovalno delo je finančno omogočilo Ministrstvo za visoko šolstvo, znanost in tehnologijo Republike Slovenije v okviru programa mladih raziskovalcev.

MREŽE PROSTE METODE NA VZPOREDNIH RAČUNALNIKIH

Marjan Šterk

Mentor: prof. dr. Borut Robič, somentor: doc. dr. Roman Trobec

Povzetek

Pričujoča doktorska disertacija obravnava numerične metode za reševanje parcialnih diferencialnih enačb, ki ne potrebujejo mreže vozlišč – mreže proste metode, s stališča njihove matematične formulacije, natančnosti, časovne zahtevnosti in možnosti uporabe na vzporednih računalnikih.

Numerično reševanje parcialnih diferencialnih enačb je sestavljeno iz pretvorbe diferencialnih enačb v sistem navadnih enačb in njegovega reševanja. Za gradnjo sistema navadnih enačb se večinoma uporabljata dve na mrežah zasnovani metodi. Metoda končnih razlik uporablja pravokotno, največkrat ekvidistantno mrežo točk. Ta metoda je hitra, enostavna in zelo dobro deluje na domenah primernih oblik, težave pa ima s poševnimi in ukrivljenimi robovi. Metoda končnih elementov razdeli domeno na mrežo majhnih likov – elementov, v dveh dimenzijah trikotnikov ali štirikotnikov. Rešitev predstavi kot linearno kombinacijo baznih funkcij, ki so definirane nad posameznimi elementi. Žal je natančnost metode zelo odvisna od kakovosti mreže elementov. Gradnja mreže je zahtevna naloga, ki navadno ne poteka povsem avtomatično, marveč zahteva delo usposobljenega človeka – strokovnjaka.

Mreže proste metode poskušajo odpraviti naštete težave. Namesto elementov jim za definicijo baznih funkcij zadošča, da po domeni razporedimo primerno število vozlišč. So manj občutljive na razporeditev vozlišč, kot metoda končnih elementov. Zaradi drugačnih baznih funkcij se spremeni tako matematična formulacija metode kot tudi njena računalniška izvedba. V nadaljevanju doktorska disertacija obravnava bazne funkcije, dobljene z aproksimacijo z gibljivimi najmanjšimi kvadrati, ki se večinoma uporabljajo pri mreže prostih metodah.

Na primeru difuzijske enačbe so podrobno razloženi vsi koraki, ki privedejo do rešitve z omenjenimi tremi metodami. Predlagan je preprost postopek za generiranje vozlišč za mreže proste metode in dva postopka za določanje nosilne domene točke. Dobljene rešitve so podrobno analizirane in primerjane s stališča natančnosti v odvisnosti od gostote mreže oziroma vozlišč. Pri mreže prostih metodah je treba poleg vozlišč izbrati še številne druge parametre, zato analiza vsebuje tudi njihovo optimizacijo za čim večjo natančnost rešitve. Dokazana je večja natančnost mreže prostih metod od metod končnih razlik in končnih elementov pri različnih oblikah domene.

Disertacija nadaljuje s podrobnim opisom predlagane izvedbe mreže prostih metod.

Ključna gradnika sta iskanje vozlišč v okolici dane točke, ki ga učinkovito reši uporaba k-D dreves, in reševanje majhnih sistemov linearnih enačb v sklopu izračuna vrednosti baznih funkcij, kjer sta primerjani dve knjižnici numeričnih metod in je izbrana hitrejša. Za predlagano izvedbo mreže prostih metod je analizirana časovna in prostorska zahtevnost in primerjana z metodama končnih razlik in končnih elementov.

Zadnji del disertacije obravnava paralelizacijo mreže prostih metod, ponovno v primerjavi z drugima dvema metodama. Predlaga dva načina delitve domene in z njo vozlišč med procesorje. Enodimenzionalna delitev razreže domeno na rezine, hierarhična pa, enako kot k-D drevo, vozlišča najprej razdeli na levo in desno polovico, nato vsako izmed njiju na spodnjo in zgornjo in tako naprej, dokler ni število delov enako številu procesorjev. Oba načina zagotavljata enakomerno obremenitev vseh procesorjev, vendar je pri hierarhični delitvi začetni zaporedni del programa nekoliko daljši. Izmerjene in analizirane so pohitritve na testnem skupku računalnikov z obema predlaganima načinoma delitve ter vpliv hitrosti komunikacije in računanja na čas izvajanja vzporednega programa.

Z opisanimi poskusi in analizami prispeva disertacija odgovore na nekatera odprta vprašanja s področja mreže prostih metod, ki jih povzame v sklepu in navede tudi možne smeri nadaljnjega raziskovanja.

MESH FREE METHODS ON PARALLEL COMPUTERS

Marjan Šterk

Supervisor: prof. dr. Borut Robič, co-supervisor: doc. dr. Roman Trobec

Abstract

Numerical methods for the solution of partial differential equations without background mesh – mesh free methods - constitute the subject of this dissertation. Particular emphasis is laid on their mathematical formulation, accuracy, time complexity, and applicability on parallel computers.

Solving partial differential equations numerically involves their conversion into a system of linear or nonlinear equations, usually algebraic, which is then solved. The finite difference and finite element methods have traditionally been used to construct the algebraic system. The finite difference method uses a rectangular, usually equidistant, mesh of points. While it is simple and fast, and works well on domains with only vertical and horizontal boundaries, any other domain shape causes a significant loss of accuracy. In the finite element method, the domain is divided into a mesh of small elements, e.g. triangles or quadrilaterals in two-dimensional problems. The approximate solution is represented as a linear combination of base functions, which are defined on individual elements. Unfortunately, mesh construction is a complex task and is not fully automated, and thus requires human effort.

The above problems can be overcome with mesh free methods, also known as meshless methods. Instead of the elements, an appropriate number of nodes are scattered throughout the domain and the base functions are defined on sets of nodes. Moreover, the solution accuracy does not depend as much on the distribution of nodes as it does in the finite element method. The different base function definition causes substantial differences in the mathematical formulation of the method as well as in the implementation on a computer. After the introductory part, this dissertation focuses on base functions constructed using the moving least squares approximation, which is the one most frequently used in mesh free methods.

The steps involved in construction of the algebraic system are shown in detail for the three methods. The diffusion equation is used as a test case. A simple node generation algorithm is proposed for the mesh free method and two algorithms for determination of the support domain of a given point. The accuracy of each of the three methods for different numbers of nodes is analysed and comparisons are made. The analysis also includes accuracy optimization of the mesh free method by finding optimal parameter values. The superior accuracy of the mesh free method over the other two methods is

shown for different domain shapes.

A detailed description of the suggested implementation of the mesh free method is given. The key idea is the use of k-D trees for finding the nodes in the vicinity of a given point. Also, two numerical libraries are tested for the solution of small linear equation systems, which constitute a part of base function evaluation. Their performances are compared and the faster is chosen. The time and space complexity of the suggested implementation are analysed and compared to the finite difference and finite element methods.

In the last part, the parallelization of the mesh free methods is discussed, again in comparison with the other two methods. Two domain decomposition methods are suggested, both of which guarantee ideal load balancing among the processors. In the one-dimensional decomposition, domain slices are assigned to each processor. The hierarchical distribution is similar to the k-D tree in the sense that it first divides the set of nodes into left and right halves, after which each is further divided into top and bottom, etc, until the number of subsets matches the number of processors. The initial, serial part of the program is somewhat longer with the hierarchical distribution. The speedups on a test cluster are measured and analysed for both decomposition methods, and the impacts of communication and computation speed are given.

The above theoretical and practical analysis contributes some new knowledge in the field of mesh free methods, which is summarized in the Conclusions, together with possible directions for future research.

Kazalo

Slike

Poglavje 1

Uvod

V uvodnem poglavju najprej predstavimo motivacijo za računalniške simulacije na različnih področjih znanosti in tehnike. Uvedemo diferencialne enačbe ter z njimi povezane oznake in terminologijo, navedemo razloge za njihovo numerično reševanje in s tem povezane težave z natančnostjo. Mreže proste metode predstavimo predvsem z vidika njihovih prednosti pred klasičnima metodama končnih razlik in končnih elementov. Poglavje zaključimo z daljšim uvodom v vzporedno računanje, kjer definiramo mere za uspešnost vzporednega računanja, naštejemo vrste vzporednih računalnikov in predstavimo model komunikacije MPI.

1.1 Pomen računalniških poskusov

Matematično *modeliranje* sistemov in njihova *simulacija* na računalnikih vse bolj dopolnjujeta tradicionalne poskuse v laboratoriju, najsi gre za naravne pojave ali umetne sisteme [HE88]. S pomočjo simulacije lahko natančno predvidimo delovanje neke naprave in drago tehnologijo izdelave uporabimo samo za različico, ki se je pri simulaciji pokazala za optimalno. Tako prihranimo čas in denar. Po drugi strani se lahko z računalniško simulacijo lotimo sicer neizvedljivih poskusov. Tako lahko simuliramo velike sisteme, na primer galaksije, ki jih v resničnem svetu ne moremo postavljati v poljubno začetno stanje; majhne sisteme, pri katerih dogajanja na atomskem nivoju ne moremo opazovati, ne da bi nanj vplivali; umetne sisteme, ki v fizičnem svetu ne morejo obstajati; poskuse, neprimerne za izvedbo, kot so npr. poskusi na ljudeh.

Povečini gre za zapletene pojave, ki jih lahko simuliramo le z nekaterimi poenostavitvami. Pri interpretaciji rezultatov se je zato vedno treba zavedati omejitev simulacije. Po drugi strani pa so takšni *računalniški poskusi* povsem ponovljivi, kar omogoča nadzorovano spreminjanje posameznega parametra. Dobljeni rezultati so tudi podrobnejši, saj lahko opazujemo časovni potek vseh spremenljivk v katerikoli točki, medtem ko pri fizičnih

poskusih uporabimo le omejeno število merilnih naprav na tistih mestih, za katera vnaprej ocenimo, da so za nas najpomembnejša.

Simulacije se rutinsko uporabljajo na mnogih področjih. Navedimo nekaj primerov, kjer uspešno nadomeščajo dražje, časovno potratnejše in včasih nevarne poskuse [FJL+88]:

- aerodinamični poskusi v vetrovniku,
- ohlajanje polizdelkov po oblikovanju (valjanju, vlečenju itd.) v kovinski industriji,
- poskusne jedrske eksplozije,
- kemijski procesi v farmacevtski industriji,
- širjenje panike v množici ljudi itd.

1.1.1 Simulacije v medicini

V medicini se za temeljne raziskave uporabljajo simulacije širjenja valovne fronte, ki nadzoruje bitje srca, prehajanje snovi skozi membrane celic itd [PHB+00]. Še večji je prispevek simulacij v aplikativnih raziskavah, saj so alternativa poskusom na živalih, zdravih ljudeh in pacientih. Tu so posebno pomembne zgoraj omenjene prednosti glede ponovljivosti in odprave težav z meritvami, vendar so tudi omejitve hujše, saj gre za zapletene in še ne povsem pojasnjene pojave na tkivih nepravilnih in spreminjajočih se oblik.

Simulacija kemoterapije za zdravljenje raka omogoča veliko hitrejše zbiranje rezultatov, na podlagi katerih se zdravnik odloči za vrsto in režim terapije [NO99]. Tudi pri iskanju načinov za zmanjšanje potrebnega števila elektrod pri elektrokardiografiji je možno večinoma uporabljati simulacijo, na realnih meritvah pa le preveriti dognanja [Avb03, AŠT02]. Manjše število elektrod poenostavi in poceni diagnostiko ter olajša meritve na nepokretnih pacientih.

Pomemben mehanizem v toplokrvnih živih bitjih, kamor med drugim spadajo vsi sesalci, je *prenos toplote* [BCW75]. Temperaturo tkiv pri nekaterih medicinskih posegih umetno spreminjamo, na primer pri zažiganju tumorjev, hlajenju srca med odprto operacijo ali hlajenju sklepov po poškodbi [TSGG98]. Razen zadnjega gre za zahtevne posege, ki so se tradicionalno izvajali ad-hoc, simulacija pa nam omogoči boljši vpogled v dogajanje in pravilnejšo izbiro parametrov terapije.

1.2 Diferencialne enačbe kot model za opisovanje naravnih in umetnih sistemov

Matematični model sistema je *množica spremenljivk*, ki podajajo *stanje sistema*, in sistem enačb, ki opisuje medsebojno odvisnost spremenljivk. Če v enačbah nastopajo odvodi

odvisnih spremenljivk, govorimo o sistemu *diferencialnih enačb* (DE), katerih rešitve so funkcije neodvisnih spremenljivk.

1.2.1 Navadne diferencialne enačbe

Sistem *navadnih diferencialnih enačb* (NDE) *prvega reda* ima eno neodvisno spremenljivko x in obliko:

$$\mathbf{F}(x, \mathbf{u}, \mathbf{u}') = 0,\ \mathbf{F} : \mathcal{R} \times \mathcal{R}^n \times \mathcal{R}^n \to \mathcal{R}^n,$$

kjer so $\mathbf{u}$ iskane funkcije:

$$\mathbf{u} = [u_1, u_2, \ldots, u_n],\ u_i : \mathcal{R} \to \mathcal{R}.$$

Sistem NDE je *ekspliciten*, če ga lahko zapišemo v obliki:

$$\mathbf{u}' = \mathbf{f}(x, \mathbf{u}),\ \mathbf{f} : \mathcal{R} \times \mathcal{R}^n \to \mathcal{R}^n. \tag{1.1}$$

Diferencialna enačba je *linearna*, če so $\mathbf{f}$ linearne funkcije. *Splošna rešitev* ene linearne NDE prvega reda je 1-parametrična družina funkcij, na primer enačba $u' = u$ ima rešitve $u = Ce^x$. Za izločitev določene *partikularne rešitve* potrebujemo dodatno informacijo. V primeru ene same enačbe zadošča, če je dan *začetni pogoj* – vrednost v neki točki $u(a) = u_a$, ki jo vstavimo v splošno rešitev in dobimo C. V primeru sistema n enačb potrebujemo n pogojev [Pol64].

Eksplicitne NDE reda n imajo obliko:

$$u^{(n)} = f(x, u, u', u'', \ldots, u^{(n-1)})$$

in jih lahko pretvorimo v sistem NDE prvega reda:

$$\begin{bmatrix} u_1' \\ u_2' \\ \vdots \\ u_{n-1}' \\ u_n' \end{bmatrix} = \begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_n \\ f(x, u_1, u_2, \ldots, u_n) \end{bmatrix}.$$

Podobno lahko sistem NDE višjega reda pretvorimo v večji sistem prvega reda [Hea02].

Primer NDE drugega reda je drugi Newtonov zakon $F = ma$, kjer iščemo odvisnost poti od časa $s(t)$, a pa je pospešek oziroma $a = s''$. Splošna rešitev je parabola

$$s(t) = \frac{F}{2m}t^2 + C_1 t + C_2,$$

za partikularno rešitev pa potrebujemo na primer dva začetna pogoja $s(a) = s_a, s'(a) = v_a$ ali dva *robna pogoja* $s(a) = s_a, s(b) = s_b$. V prvem primeru govorimo o začetnem, v drugem pa o robnem problemu. V splošnem za sistem n linearnih diferencialnih enačb redov r_i potrebujemo $\sum_{i=1}^{n} r_i$ pogojev.

1.2.2 Parcialne diferencialne enačbe

Pogosto matematični model sistema vsebuje dve ali več neodvisnih spremenljivk, na primer pri fizikalnih pojavih največ tri prostorske in največ eno časovno koordinato. V tem primeru dobimo *parcialne diferencialne enačbe* (PDE), saj v njih nastopajo parcialni odvodi [Pol64].

Primer PDE je enodimenzijska *difuzijska enačba*:

$$\frac{\partial u}{\partial t} - c\frac{\partial^2 u}{\partial x^2} - g = 0, \tag{1.2}$$

s katero opišemo različne pojave, na primer prenos toplote v trdni snovi – v tem primeru rešitev $u(x,t)$ predstavlja temperaturo [Ö94].

Robni in začetni pogoji

Navadno nas zanima rešitev PDE znotraj neke *domene* Ω v prostoru, za kar potrebujemo robne pogoje na robovih $\Gamma = \partial\Omega$ domene Ω. Ti so v splošnem podani s kombinacijo vrednosti rešitve in njenih odvodov, izmed katerih navadno nastopajo le prvi odvodi v smeri normale na rob domene [Liu03]:

$$\begin{aligned} u &= \overline{u},\ \mathbf{x} \in \Gamma_u \text{ (\textit{bistveni} (angl. essential) ali \textit{Dirichletovi} robni pogoji)},\\ \frac{\partial u}{\partial n} &= \overline{n},\ \mathbf{x} \in \Gamma_n \text{ (\textit{naravni} (angl. natural) ali \textit{Neumannovi} robni pogoji)},\\ \Gamma &= \Gamma_u \cup \Gamma_n. \end{aligned}$$

V primeru *časovno odvisnih* PDE potrebujemo še začetne pogoje $u(\mathbf{x}, t_0) = u_0$, kar nam omogoči izračun vrednosti rešitve za katerikoli poznejši čas $t > t_0$, teoretično pa tudi za predhodne čase $t < t_0$. V primeru enačbe 1.2 je domena interval $[a, b]$, primer robnih pogojev pa:

$$u(a,t) = u_a,\ \left.\frac{\partial u}{\partial x}\right|_{x=b} = q_b\ ;\quad \Gamma_u = \{a\}, \Gamma_n = \{b\}.$$

Kot bomo videli pozneje, je posebno hud zaplet, če se poleg stanja sistema s časom spreminja tudi oblika ali velikost domene.

1.3 Numerično računanje

Za večino diferencialnih enačb, ki opisujejo simulirane sisteme, *analitična rešitev* bodisi:

- ne obstaja v smislu, da bi jo lahko izrazili z elementarnimi funkcijami,
- je ne znamo najti

- ali pa jo znamo najti le za nekatere pravilne oblike domene in posebno enostavne robne in začetne pogoje. Tak primer je dvodimenzijska difuzijska enačba na kvadratni domeni s konstantnimi začetnimi pogoji in Dirichletovimi robnimi pogoji $\overline{u} = 0$ na vseh štirih robovih [GC98].

Zato se moramo odpovedati splošnosti in sistem rešiti *numerično* samo za konkretni primer ali družino primerov.

Pri reševanju PDE moramo najprej *diskretizirati domeno* Ω z ustrezno *numerično metodo*, na primer *metodo končnih razlik*. Pri časovno neodvisnih PDE s tem dobimo *sistem navadnih enačb*, pri časovno odvisnih pa sistem NDE, ki ga z *diskretizacijo časa* prav tako pretvorimo v sistem navadnih enačb. Če in samo če sta PDE in diskretizacija linearni, je dobljeni končni sistem linearen. Tudi slednjega rešimo numerično [Hea02].

1.3.1 Natančnost in stabilnost

Poleg splošnosti se z numeričnim reševanjem odpovemo tudi *točnosti* rešitve. Pri mnogih metodah začetni približek rešitve skozi *iteracije konvergira* k točni rešitvi. Računanje ustavimo, ko se pravi vrednosti dovolj približamo. Vendar rezultat ne bi bil točen niti, če bi lahko na primer sešteli neskončno členov vsote, ker so realna števila v računalniku običajno predstavljena s *plavajočo vejico* s 53-mestno dvojiško natančnostjo, kar ustreza približno 16 desetiškim mestom [Kod00], v vsakem primeru pa je natančnost omejena in rezultat vsake aritmetične operacije zaokrožen [Hea02]. Natančneje povedano, število 1 in prvo naslednje predstavljivo število se razlikujeta za strojno natančnost $\epsilon_{mach} \approx 2.2 \cdot 10^{-16}$. To ima poleg napake na 16. decimalnem mestu bolj temeljne posledice, med katerimi omenimo izgubo *asociativnosti operacij* in *relacije enakosti*. Tako velja na primer:

$$(1 + (-1)) + 10^{-20} = 10^{-20} \neq 1 + ((-1) + 10^{-20}) = 0.$$

Prav tako ne smemo uporabljati pogoja $f(x) \stackrel{?}{=} 0$, ki zaradi napake morda ne bo izpolnjen, čeprav bi moral biti, marveč $|f(x)| \stackrel{?}{<} \epsilon$, kjer moramo ϵ izbrati smiselno glede na pričakovane vrednosti $f(x)$.

Računske napake lahko razdelimo na [Ore97]:

- *zaokrožitvene napake*, ki jih ne bi bilo, če bi bile predstavitev realnih števil in aritmetične operacije neskončno natančne,
- *napake numeričnih metod*, ki nastanejo na primer z zamenjavo neskončne vsote z vsoto prvih nekaj členov ali z diskretizacijo zvezne domene Ω s končnim številom točk.

Računanje pa ni edini vir napak. Lahko že model ne ustreza dejanskemu sistemu, ker smo ga z namenom lažjega reševanja preveč poenostavili ali pa sistema celo ne znamo pravilno opisati. Vhodni podatki, ki so rezultat empiričnih meritev ali predhodnih izračunov,

imajo prav tako omejeno natančnost. Pomemben pojem je *pogojenost* (angl. conditioning) sistema, ki pomeni občutljivost na vhodne podatke. Slabo pogojen sistem je kaotičen, saj se že pri majhni spremembi vhodnih podatkov obnaša bistveno drugače.

Pojem *stabilnosti* numerične metode je analogen pojmu pogojenosti sistema – (ne)občutljivost rezultata na motnje, ki so posledica napak v vhodnih podatkih in zaokrožitvenih napak. Stabilnost pa še ne zagotavlja *natančnosti* (bližine rezultata točni rešitvi problema), če je simulirani sistem slabo pogojen. Slabe rezultate dobimo tako zaradi nestabilnosti algoritma kot zaradi slabe pogojenosti sistema [GO93, Hea02].

1.3.2 Mere za natančnost

Za primerjavo med numeričnimi metodami potrebujemo kvantitativno oceno natančnosti. Če je približna rešitev vektor $\hat{\mathbf{u}}$ ali funkcija $\hat{u}(\mathbf{x})$ in poznamo točno rešitev, lahko definiramo *absolutno*:

$$\mathbf{e} = \hat{\mathbf{u}} - \mathbf{u},\ e(\mathbf{x}) = \hat{u}(\mathbf{x}) - u(\mathbf{x})$$

in *relativno napako*:

$$\mathbf{e}_r = \frac{\hat{\mathbf{u}} - \mathbf{u}}{||\mathbf{u}||},\ e_r(\mathbf{x}) = \frac{\hat{u}(\mathbf{x}) - u(\mathbf{x})}{u(\mathbf{x})},$$

kjer je $||\mathbf{u}||$ izbrana vektorska norma, na primer *evklidska* norma $||\mathbf{u}||_2$ ali *neskončna* norma $||\mathbf{u}||_\infty$, imenovana tudi *max-norma*:

$$||\mathbf{u}||_2 = \sqrt{\sum_i u_i^2}\ ,\quad ||\mathbf{u}||_\infty = \max_i |u_i|.$$

Relativna napaka je seveda uporabna le tam, kjer je točna rešitev različna od 0. Zaželena lastnost numerične metode je, da nam poleg približne rešitve da tudi oceno napake, na primer njeno zgornjo mejo [Hea02].

Numerično računanje dejansko uporabljamo tam, kjer točne rešitve ne poznamo. Pri metodah, ki ne dajo ocene napake, je edina uporabna mera *ostanek* (angl. residual) – razlika med levo in desno stranjo enačbe, ko vanjo vstavimo približno rešitev. V primerih sistema linearnih enačb $A\mathbf{x} = \mathbf{b}$ in PDE $F(\mathbf{x}, u, u_{x_1}, u_{x_2}, u_{x_1x_2}, \ldots) = 0$ je ostanek:

$$\mathbf{r} = \mathbf{b} - A\hat{\mathbf{x}} \qquad \text{in} \qquad r(\mathbf{x}) = F(\mathbf{x}, \hat{u}, \hat{u}_{x_1}, \ldots),$$

kjer smo označili:

$$u_{x_1} = \frac{\partial u(\mathbf{x})}{\partial x_1},\ u_{x_1x_2} = \frac{\partial^2 u(\mathbf{x})}{\partial x_1 \partial x_2}\ \text{ itd.}$$

Pri dobro pogojenih sistemih nam majhen ostanek zagotavlja tudi majhno napako, pri slabo pogojenih pa o napaki ne pove dosti [GO93].

V primeru vektorjev lahko skalarno mero ostanka dobimo z uporabo vektorske norme, v primeru funkcij pa s funkcijsko normo, na primer integralom po celotni domeni ali uteženim integralom po delu domene:

$$r = \int_{\Omega} |r(\mathbf{x})| d\Omega \qquad \text{ali} \qquad r = \int_{\Omega_Q} W(\mathbf{x}) |r(\mathbf{x})| d\Omega_Q,$$

kjer je $\Omega_Q \subset \Omega$ in $W(\mathbf{x}) > 0$ *utežna funkcija*. Pri linearnih sistemih je koristna mera tudi *relativni ostanek*:

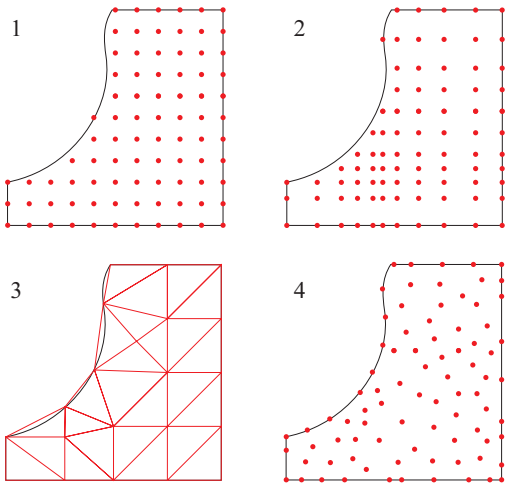
$$\mathbf{r}_r = \frac{\mathbf{b} - A\hat{\mathbf{x}}}{||\mathbf{b}||}.$$

1.4 Mreže proste metode

Za diskretizacijo domene in pretvorbo PDE v sistem navadnih enačb obstaja množica metod [Liu03]. Domeno lahko predstavimo na primer s:

1. točkami v ekvidistantni mreži, ki jih bomo imenovali *vozlišča* (angl. nodes),
2. vozlišči v mreži spremenljive gostote,
3. enostavnimi liki, ki zapolnijo ravnino (v primeru dveh dimenzij), oziroma enostavnimi telesi, ki zapolnijo prostor (v primeru treh dimenzij),
4. nepravilno raztresenimi vozlišči.

Naštete možnosti so prikazane na sliki 1.1. Očitno je, da enostavnejša diskretizacija slabše opiše domeno, vendar močno olajša formulacijo numerične metode na tako diskretizirani domeni. Omenimo še, da so v eni dimenziji zadnje tri možnosti enakovredne.



Slika 1.1: Nekateri načini diskretizacije domene.

Približek zvezne rešitve $u(\mathbf{x}), \mathbf{x} \in \Omega$ lahko definiramo kot vektor $\mathbf{u}$ vrednosti približka v posameznih vozliščih, medtem ko vrednosti med vozlišči niso definirane. Približek rešitve

lahko definiramo tudi kot linearno kombinacijo *baznih funkcij* (angl. base functions) ϕ:

$$\hat{u}(\mathbf{x}) = \sum_j u_j \phi_j(\mathbf{x}),$$

kjer je $\mathbf{u} = \{u_j\}$ vektor *koeficientov* oziroma *parametrov*, pri čemer je približek $\hat{u}(\mathbf{x})$ zdaj definiran na celi domeni [Hea02]. Reševanje PDE se v obeh primerih prevede na reševanje sistema navadnih enačb, v katerem kot neznanka nastopa $\mathbf{u}$.

1.4.1 Slabosti na mreži zasnovanih metod

V standardnih metodah za reševanje PDE, kot sta metodi *končnih razlik* (angl. finite difference method) [Ö94] in *končnih elementov* (angl. finite element method) [Kat03], je bistvena lastnost diskretizacije mrežna povezanost oziroma *sosednost* vozlišč. Diskretizacijo lahko predstavimo z grafom, katerega vozlišča so vozlišča pravilne mreže oziroma oglišča likov, povezave pa relacija sosednosti vozlišč oziroma stranice likov. Poglejmo, kakšne težave prinaša takšna togost.

Metoda končnih razlik mora za vzpostavitev sosednosti, ki je potrebna pri pretvorbi PDE v sistem navadnih enačb, uporabljati vozlišča na pravilni mreži, ki slabo opisuje poševne robove in sploh ne more opisati spreminjanja oblike domene. Čeprav je vozlišča zelo enostavno generirati, je za obravnavo spremenljivih oblik potrebno uporabiti posebne prijeme, kar to prednost izniči. Kljub vsemu se metoda končnih razlik veliko uporablja, še posebno za simulacije toka tekočin, raziskujejo pa se tudi možnosti uporabe na nepravilni mreži vozlišč [Liu03].

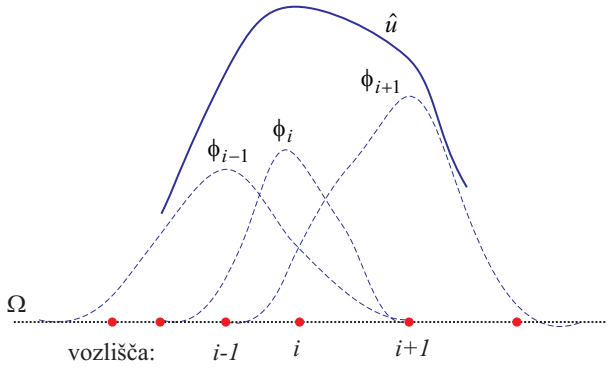
Metoda končnih elementov domeno diskretizira z liki, ki jih iz zgodovinskih razlogov imenuje *elementi* [Buc95]. Elementi lahko bolje opišejo obliko domene kot mreža ekvidistantnih točk, poleg tega metoda končnih elementov predstavi približek z baznimi funkcijami, kar tudi prispeva k večji natančnosti od metode končnih razlik. Elementi ne smejo biti niti preveliki niti preveč *izrojeni*, sicer bo napaka velika. Relacija sosednosti je potrebna tako pri definiciji baznih funkcij kot tudi pri pretvorbi v sistem navadnih enačb.

Kljub množici raziskav še vedno ni znan popolnoma avtomatski in robusten postopek pretvorbe tridimenzionalnih teles v mrežo elementov, zato to opravilo zahteva sodelovanje visoko usposobljenega strokovnjaka [FM04, IOCP03]. Niso redki primeri, ko je za izdelavo mreže potrebno za cel red velikosti več časa in denarja kot za vse ostale korake poskusa skupaj [WLSO01, BE95]. Najbolj do izraza ta slabost pride, če se domena in z njo mreža elementov spreminjata. Prej ali slej bodo elementi na delu domene postali preveliki ali preveč izrojeni, zato bo potrebno novo mrežo izdelati večkrat v toku simulacije. Poleg tega ima metoda končnih elementov težave z nekaterimi pojavi, na primer nastajanjem razpok v materialu v simulacijah mehanike loma, ki vedno sledijo robovom elementov, kar seveda ni realno [Liu03].

1.4.2 Odprava mreže

V začetku devetdesetih let so se pojavile *mreže proste metode* (MPM, angl. mesh free methods ali meshless methods, slov. tudi brezmrežne metode), s katerimi se lahko izognemo zgoraj opisanim težavam [FM04, AS02, BKO+96]. Namesto diskretizacije domene z v mrežo povezanimi vozlišči domeno predstavimo z vozlišči, razmeščenimi po robovih in notranjosti domene. Mreže, ki bi predpisovala relacijo sosednosti med vozlišči, ni več, zato sosednost vsakič sproti izračunamo le iz razdalje med vozlišči. Ta lastnost omogoča tudi poljubno premikanje, dodajanje in odvzemanje vozlišč v toku simulacije, saj bo sosednost avtomatično sledila spremembam položajev vozlišč. Vozlišča lahko generiramo z avtomatičnim postopkom, ki mora upoštevati le nekatere enostavne omejitve, na primer dano gostoto vozlišč na robovih in v notranjosti ter potrebno minimalno razdaljo med dvema vozliščema.

Približek rešitve definiramo kot linearno kombinacijo baznih funkcij z *lokalnim nosilcem* (angl. local support) – funkcija $\phi_i(\mathbf{x})$ ima neničelne vrednosti le v točkah $\mathbf{x}$, ki ležijo v bližini vozlišča i. Primer enodimenzionalne domene z vozlišči, pripadajočimi baznimi funkcijami ϕ in njihovo linearno kombinacijo $\hat{u} = \sum_j \phi_j$ prikazuje slika 1.2.



Slika 1.2: Predstavitev enodimenzionalne domene z vozlišči (rdeče točke) in baznimi funkcijami MPM.

1.4.3 Različice mreže prostih metod

Zaradi bliskovitega razvoja so se MPM razvile v nepregledno množico različic, katerih večino lahko uvrstimo v naslednje razrede:

- Metode, ki zvezne količine predstavijo z *zglajenimi delci* (angl. smoothed particle methods) in PDE pretvorijo v odnose med njimi [Luc77]. Zelo uporabne so za simulacijo nekaterih pojavov, denimo toka tekočine. Problematično je vsiljevanje robnih pogojev, saj ni sistematičnega načina za obravnavo robov [FM04].

- Metode, ki še vedno zahtevajo mrežo *elementov v ozadju* (angl. background mesh), s pomočjo katere izvajajo *numerično integracijo*, potrebno pri pretvorbi PDE v sistem navadnih enačb. To mrežo je lažje generirati, saj ni potrebna za definicijo baznih funkcij in zato ni povezana z vozlišči, s katerimi predstavimo domeno. Primer takšne metode je *elementov prosta metoda Galerkina* (EPG, angl. element free Galerkin method, EFG) [BLG94].
- Metode, ki se integraciji izognejo, na primer *kolokacijske MPM* in metode končnih razlik na nepravilni mreži. Huda slabost teh metod je občutljivost na postavitev vozlišč [AS02].
- Metode, ki numerično integracijo izvajajo lokalno – za vsako vozlišče integrirajo po njegovi okolici, ki jo imenujejo *integracijska* ali *kvadraturna domena* (angl. integration domain, quadrature domain). Tako morajo vsakič generirati zgolj trivialno lokalno mrežo, vendar lokalna integracija poveča računsko zahtevnost metode. V ta razred sodijo *mreže proste lokalne Petrov-Galerkinove* metode (MLPG, angl. meshless local Petrov-Galerkin methods), s katerimi se v tem delu največ ukvarjamo [AS02].

Idealna mreže prosta metoda bi se numerični integraciji izognila z uporabo takšnih baznih funkcij in takšnih oblik integracijskih domen, da bi jih bilo moč integrirati analitično. Zaenkrat ni znana nobena takšna metoda [Liu03].

1.5 Vzporedno računanje

Z razvojem v znanosti, tehniki, medicini in drugje se pojavljajo potrebe po vse zahtevnejših izračunih nad vse večjimi množicami podatkov, pa naj bo to simulacija prenosa toplote v človeškem organu, izračun vremena za naslednji dan s prostorsko ločljivostjo enega kilometra ali pa tekmovanje v računanju čimveč decimalk števila π. Poleg tega uporaba nekaterih novih postopkov in numeričnih metod, kot so na primer MPM, razbremeni človeka na račun večje računske zahtevnosti. Bliskovitemu razvoju strojne opreme navkljub so izračuni prepočasni, zato problem razdelimo na čimbolj neodvisne podprobleme. V računanje vprežemo več računalnikov, ki podprobleme rešujejo hkrati – *vzporedno*, pri tem pa po potrebi komunicirajo med seboj [FJL+88].

1.5.1 Uspešnost vzporednega računanja

Uspešnost vzporednega računanja je odvisna od narave problema. V splošnem velja: čim manj časa se porabi za komuniciranje med posameznimi računalniki, tem več pridobimo s prehodom na vzporedno računanje in tem več računalnikov je smiselno uporabiti. Čas komunikacije namreč v najboljšem primeru pada počasneje, kot raste število računalnikov, pogosto ostaja enak, pri nekaterih problemih pa celo raste [FJL+88]. Iz Amdahlovega

zakona [Kod00] zato sledi, da pri preveč računalnikih dosežemo mejo, ko kljub večji skupni računski moči zaradi povečanega deleža komuniciranja hitrost ne raste več bistveno oziroma celo pada.

Pri računanju števila π je treba seštevati člene nekega eksplicitno podanega zaporedja, ki ga lahko razdelimo na toliko delov, kolikor imamo računalnikov, na koncu pa le seštejemo delne rezultate. Potrebujemo minimalno količino komunikacije. Simulacija prenosa toplote zahteva več komunikacije, pa tudi izvedba je zahtevnejša. Dogajanje v neki točki vpliva na sosednje točke, prek njih pa posredno tudi na vse ostale [GO93].

Omenimo dve uveljavljeni količini, s katerima ocenjujemo kakovost vzporednih programov [GGKK03]. *Pohitritev* (angl. speedup) je razmerje med časom izvajanja programa na enem računalniku in časom izvajanja na vzporednem sistemu. *Učinkovitost* (angl. efficiency) je razmerje med pohitritvijo in številom računalnikov v vzporednem sistemu. Teoretično je idealna učinkovitost 1, torej p-krat hitrejše izvajanje na p računalnikih oziroma *linearna pohitritev*. V praksi lahko zaradi vplivov predpomnilnika izmerimo tudi višjo pohitritev, kajti vsi računalniki skupaj imajo več predpomnilnika, zato je v vzporednem programu verjetnost predpomnilniškega zadetka večja. Primerjava z enim računalnikom s p-krat večjim predpomnilnikom bi seveda zopet pokazala pohitritev, manjšo od p.

1.5.2 Vrste vzporednih računalnikov

Vzporedni računalniki se delijo glede na način komuniciranja med procesorji. Ti lahko komunicirajo preko *skupnega pomnilnika* (angl. shared memory), pri čemer si lahko delijo ves pomnilnik ali le njegov del, lahko pa si pošiljajo *sporočila* preko računalniškega omrežja [GGKK03, AG94].

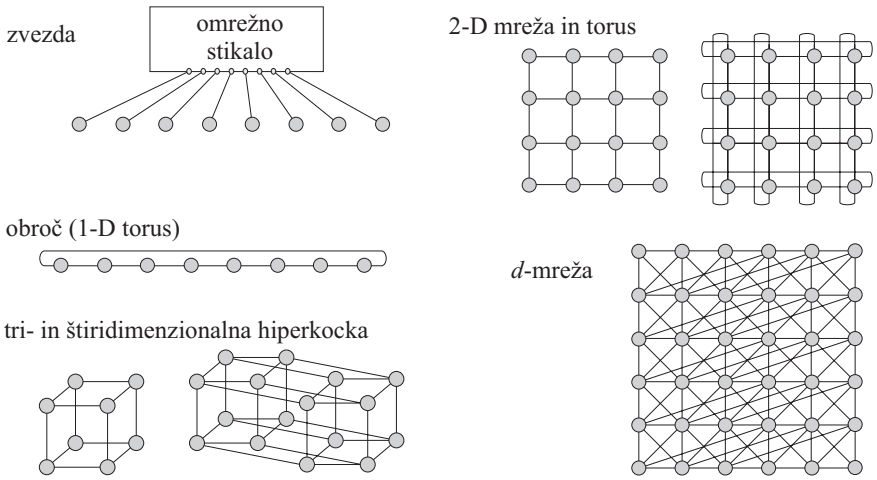
Simetrični večprocesorski računalniki (angl. symmetric multiprocessors, SMP) so dandanes najbolj razširjena vrsta vzporednih računalnikov z deljenim pomnilnikom. Uporabljajo standardne komponente in operacijske sisteme [GGKK03]. Vsi procesorji so lahko na skupnem vodilu, kjer je tudi ves pomnilnik (takšni so sistemi s procesorji Intel xeon), ali pa je pomnilnik nameščen lokalno pri procesorjih, ki so povezani s hitrimi neposrednimi povezavami (sistemi s procesorji AMD opteron). V vsakem primeru lahko vsi procesorji dostopajo do celotnega pomnilnika, vendar je pri drugi možnosti dostop do lokalnega pomnilnika hitrejši kot do ostalih delov. Prednosti takšnih računalnikov sta hitra komunikacija in ugodna cena, saj se ne podvajajo ostale komponente sistema (vhodno-izhodni krmilniki, diski itd). Kljub ozkemu grlu pri dostopu do pomnilnika postajajo vse bolj priljubljeni, pojavljajo se tudi *večjedrni čipi* (angl. multicore chips) – več procesorjev na istem čipu.

Redkejši in dražji so namensko zgrajeni vzporedni računalniki, kakršen je NEC Earth Simulator. Navadno vsebujejo osnovne enote z nekaj procesorji z deljenim pomnilnikom. Komunikaciji med enotami je namenjena posebej zasnovana povezovalna mreža, preko

katere si procesorji pošiljajo sporočila ali dostopajo do pomnilnika v drugih enotah. Skupna pasovna širina pomnilnika raste z dodajanjem enot. Tovrstni računalniki imajo lahko več procesorjev, tudi do nekaj tisoč [AG94].

Po letu 1990 se je pojavila nova vrsta vzporednih računalnikov, množica eno- ali večprocesorskih osebnih računalnikov ali delovnih postaj, povezanih v lokalno omrežje – *skupek* ali *gručo* (angl. cluster), ki združuje nizko ceno in zmožnost uporabe velikega števila procesorjev [MR00, Pip00]. V zadnjem času je tako zgrajeno tudi veliko superračunalnikov, na primer družina ASCI, ki ima tudi več kot 10.000 procesorjev.

Povezovalno omrežje ima lahko obliko zvezde z omrežnim stikalom (angl. switch) v sredini ali pa so nekateri računalniki neposredno povezani – nekaj možnih povezovalnih topologij prikazuje slika 1.3. Najpogostejša je uporaba omrežnega stikala, preko katerega



Slika 1.3: Nekateri načini povezave računalnikov v skupek.

lahko v primeru polno zmogljivega stikala hkrati komunicira $p/2$ parov procesorjev, noben računalnik pa ne more komunicirati z dvema drugima hkrati, saj ima le eno mrežno kartico. Druge topologije lahko to omejitev odpravijo [GGKK03, AG94].

Poceni in za mnoge aplikacije še dovolj zmogljiva tehnologija povezovanja je gigabitni Ethernet, obstaja pa tudi več dražjih in hitrejših alternativ, kot sta Myrinet in SCI [MR00]. Prednost slednjih je predvsem manjša zakasnitev pri kratkih sporočilih.

1.5.3 Standard MPI

Standard *MPI* (Message Passing Interface, slovensko vmesnik za izmenjavo sporočil) [SOHL+96] predpisuje nabor funkcij za programiranje vzporednih računalnikov, ki komunicirajo s sporočili. Z njegovo pomočjo uporabnik programira komunikacijo med procesi na višjem nivoju, kakor če bi uporabljal sistemske funkcije, poleg tega pa so programi

prenosljivi med različnimi operacijskimi sistemi. Vsi računalniki so iz MPI vidni enako, ne glede na število procesorjev in vrsto vzporednosti, pa naj gre za dvoprocesorski osebni računalnik, skupek delovnih postaj ali pa večprocesorski superračunalnik. Poleg tega je zagotovljena zanesljivost komuniciranja. MPI lahko uporabljamo v jezikih fortran in C/C++, nekatere izvedbe pa tudi v javi.

Osnovna enota programiranja z MPI je *proces*. Na vsakem računalniku lahko teče eden ali več procesov, ne glede na to, koliko procesorjev ima računalnik. To omogoča tako uporabo skupkov večprocesorskih računalnikov kot tudi razvoj in preizkušanje programov na enoprocesorskem računalniku. Vsak proces dobi enolično *identifikacijsko številko*, s katero se ga naslavlja.

Osnovno komunikacijo predstavlja pošiljanje in prejemanje sporočil, čemur je namenjen par funkcij `MPI_Send` in `MPI_Recv`. `MPI_Bcast` razpošlje enako sporočilo vsem procesom, `MPI_Reduce` pa zbere rezultate, na primer vsoto ali minimum sporočil z vseh procesov na enem. Tipičen vzporeden program uporablja nekaj deset različnih funkcij MPI, čeprav jih standard vsebuje veliko več.

Za namensko zgrajene vzporedne računalnike navadno proizvajalec priskrbi tudi prirejeno, optimizirano izvedbo standarda MPI. Za ostale uporabimo katero od prenosljivih, navadno brezplačnih izvedb, na primer mpich [MPI] ali LAM/MPI [LAM]. Posamezne izvedbe se v odvisnosti od uporabljene strojne opreme lahko zelo razlikujejo v zmogljivosti, zato je priporočljivo pred odločitvijo preizkusiti več različic [Pip00, RTŠ05b].

Poglavje 2

Numerično reševanje parcialnih diferencialnih enačb

V tem poglavju najprej opišemo principe, uporabne za prostorsko in časovno diskretizacijo. Podrobneje opišemo difuzijsko enačbo in podamo primer problema, za katerega poznamo analitično rešitev in nam bo služil za preverjanje delovanja vseh metod. Nato se posvetimo klasičnima metodama končnih razlik in končnih elementov. Z obema rešimo zelo enostavno diferencialno enačbo in ju nato razvijemo do uporabnosti na dvodimenzijski difuzijski enačbi. Na koncu bralca uvedemo še v problem reševanja končnega sistema linearnih enačb, ki ga v tem delu sicer ne analiziramo zelo podrobno, vendar se ga moramo vseskozi zavedati.

2.1 Prostorska diskretizacija

Rešitev PDE je funkcija $u(\mathbf{x})$, ki je v računalniku ne moremo predstaviti drugače kot diskretno, s končnim vektorjem parametrov $\mathbf{u} = (u_1, \ldots, u_n)$. Način predstavitve določa tudi predstavitev vseh v enačbi nastopajočih prostorskih odvodov, ki jo prav tako potrebujemo.

2.1.1 Kolokacija

Če znamo odvode, ki nastopajo v časovno neodvisni PDE $F(\mathbf{x}, u, \ldots) = 0$, izraziti z $\mathbf{u}$, lahko tvorimo sistem navadnih enačb, ki zahtevajo, da je PDE izpolnjena točno v N izbranih točkah – *vozliščih* (angl. nodes):

$$F(\mathbf{x}_i, u_i(\mathbf{x}_i), \ldots) = 0, \ i = 1 \ldots N.$$

Z drugimi besedami, enačbe zahtevajo, da je ostanek v vozliščih enak 0. Temu postopku pravimo *kolokacija* (angl. collocation) v vozliščih [Hea02].

Zelo pomembno se je zavedati, da vrednost s kolokacijo dobljene rešitve v vozliščih ne bo točna in napaka tam ne bo 0. Ilustrirajmo to na primeru enačbe z robnimi pogoji:

$$u''(x) = 12x^2,\ u(0) = u(1) = 0, \tag{2.1}$$

katere točna rešitev je $u(x) = x^4 - x$. Približno rešitev bomo predstavili z vektorjem vrednosti v ekvidistantnih vozliščih: $\mathbf{u} = [\hat{u}(0), \hat{u}(\frac{1}{2}), \hat{u}(1)]^{\mathrm{T}}$. V robnih vozliščih kolokacija zahteva izpolnjevanje robnih pogojev, v notranjem vozlišču pa izpolnjevanje diferencialne enačbe. Potrebni približek drugega odvoda dobimo iz znanih formul, izpeljanih iz Taylorjeve vrste [Ore97]:

$$\begin{aligned} u'(x) &\approx \frac{u(x+\Delta x) - u(x-\Delta x)}{2\Delta x}, \\ u''(x) &\approx \frac{u(x-\Delta x) - 2u(x) + u(x+\Delta x)}{(\Delta x)^2}, \end{aligned} \tag{2.2}$$

kjer je Δx razdalja med vozlišči v smeri x. Dobimo linearni sistem:

$$\begin{aligned} \hat{u}(0) &= 0, \\ \frac{\hat{u}(0) - 2\hat{u}(\frac{1}{2}) + \hat{u}(1)}{(\frac{1}{2})^2} &= 3, \\ \hat{u}(1) &= 0, \end{aligned}$$

katerega rešitev je v robnih vozliščih točna, v notranjem vozlišču pa $\hat{u}(\frac{1}{2}) = -\frac{3}{8}$, medtem ko je prava vrednost $-\frac{7}{16}$. Natančnejšo rešitev bi dobili z uporabo več vozlišč.

2.1.2 Uteženi ostanek

Pri drugi možnosti diskretizacije je $\mathbf{u}$ vektor koeficientov linearne kombinacije baznih funkcij, ki predstavlja približek $\hat{u}$:

$$\hat{u}(\mathbf{x}) = \sum_j u_j \phi_j(\mathbf{x}) = \mathbf{u}^{\mathrm{T}} \mathbf{\Phi}(\mathbf{x}).$$

Tako definirani približni rešitvi pravimo tudi *poskusna funkcija* (angl. trial function) [Buc95]. Za odvajanje $\hat{u}$ moramo poznati odvode baznih funkcij:

$$\frac{\partial \hat{u}}{\partial x_i} = \sum_j u_j \frac{\partial \phi_j}{\partial x_i} = \mathbf{u}^{\mathrm{T}} \mathbf{\Phi}_{x_i}$$

in podobno za višje in mešane odvode, zato morajo biti bazne funkcije dovoljkrat odvedljive.

Spet lahko uporabimo kolokacijo, vendar ta daje slabše rešitve, ker zahteva ujemanje s PDE le v vozliščih, poleg tega pa je zelo občutljiva na razporeditev vozlišč [FM04]. Navadne enačbe raje konstruiramo tako, da vsaka izmed njih zahteva neke vrste minimizacijo ostanka na določenem področju. Diferencialna enačba $F(\mathbf{x}, u, \ldots) = 0$ je ekvivalentna neskončnemu sistemu enačb, ki zahteva, naj bo ostanek *ortogonalen* na vsako realno funkcijo [Pol64]:

$$\int_\Omega W(\mathbf{x})F(\mathbf{x}, u, \ldots)d\Omega = 0 \text{ za poljubno realno funkcijo } W(\mathbf{x}). \tag{2.3}$$

Izberemo toliko *uteženih funkcij* (angl. weight functions) W_i, ki jim pravimo tudi *testne funkcije*, kolikor parametrov ima poskusna funkcija:

$$\int_\Omega W_i(\mathbf{x})r(\mathbf{x})d\Omega = \int_\Omega W_i(\mathbf{x})F(\mathbf{x}, \mathbf{u}^{\mathrm{T}}\mathbf{\Phi}(\mathbf{x}), \ldots)d\Omega = 0. \tag{2.4}$$

Če so funkcije W in ϕ dovolj enostavne, integral izrazimo analitično, sicer pa ga izračunamo numerično. Dobimo sistem enačb, ki mu pravimo *šibka oblika* PDE (angl. weak form). Njegove neznanke so koeficienti poskusne funkcije u_j, rešitev pa funkcija, ki je blizu pravi rešitvi. Iz časovno neodvisne PDE dobimo sistem navadnih enačb in konstantne koeficiente u_j. V nasprotnem primeru so $u_j(t)$ odvisni od časa in dobimo sistem NDE, ki ga rešimo z diskretizacijo časa [Hea02]. Z uteženim ostankom bomo dobili točno rešitev vedno, kadar jo je moč izraziti z uporabljenimi baznimi funkcijami [Liu03], kar pride še posebej prav pri testiranju metod.

Testne funkcije lahko načeloma izbiramo poljubno, vendar izbira do neke mere vpliva na natančnost končne rešitve – na primer izbira $W_i = 0$ očitno ne bo dala dobrih rezultatov. Največkrat se uporabi bodisi funkcije, konstantne na neki poddomeni $\Omega_{Qi} \subseteq \Omega$, ali pa funkcije v obliki *klobuka* (angl. hat functions) z vrhovi v različnih točkah. Pogosto so to kar bazne funkcije $\mathbf{W} = \mathbf{\Phi}$. V tem primeru se uporablja ime metoda *Galerkina*, če so različne, pa metoda *Petrov-Galerkina* [Liu03]. Če za W_i izberemo *Diracove δ funkcije* z vrhovi v vozliščih:

$$W_i(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_i) = \begin{cases} \infty & \mathbf{x} = \mathbf{x}_i \\ 0 & \text{sicer,} \end{cases}$$

je integral (2.4) enak vrednosti ostanka v vozlišču in dobimo kolokacijo [Hea02].

Numerična integracija

Kadar integrala v (2.4) ne znamo izračunati analitično, se v metodah končnih elementov in mreže prostih metodah največ uporablja *Gaussova kvadraturna formula* [Hea02]. Njena splošna oblika je:

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} f(x_i)w_i.$$

Abscise x_i in pripadajoče uteži w_i dobimo iz sistema enačb, ki zahteva, naj formula točno integrira poljuben polinom z enakim številom parametrov kot je število neznank v tem sistemu, to je polinom stopnje $2n-1$ z $2n$ parametri. Sistem je nelinearen in težaven za reševanje, zato pri izbranem n vrednosti x_i in w_i izračunamo enkrat in jih tabeliramo. Tabelirane vrednosti lahko najdemo na primer v [AS76].

Če neka metoda za numerično integracijo točno integrira poljuben polinom stopnje d, obstaja pa polinom stopnje $d+1$, ki ga ne integrira točno, pravimo, da je metoda *reda* (angl. order) $d+1$. Gaussova formula na n točkah je torej reda $2n$ [GO93].

Formula zahteva linearno transformacijo funkcije z integracijskega intervala $[a,b]$ na interval $[-1,1]$ in izračun funkcijske vrednosti v n točkah. Napaka je sorazmerna $(b-a)^{2n}$. Če potrebujemo natančnejši izračun, lahko vzamemo višjo stopnjo formule ali pa interval razdelimo na več manjših in integriramo vsakega posebej.

V več dimenzijah je domena $\Omega_{Qi} = \{\mathbf{x}|W_i(\mathbf{x}) \neq 0\}$, po kateri integriramo integral iz enačbe 2.4, običajno pravilne oblike (pravokotnik, trikotnik, krog oziroma tovrstna telesa v treh dimenzijah) ali pa jo lahko razdelimo na takšne like. Po pravokotniku integriramo enostavno tako, da za abscise vzamemo kartezični produkt $X_Q \times X_Q$, za uteži pa $\mathbf{w}\mathbf{w}^{\mathrm{T}}$, kjer sta $X_Q = \{x_i\}$ množica abscis in $\mathbf{w}$ vektor uteži enodimenzionalne formule. Za ostale pravilne oblike lahko abscise in uteži izpeljemo z zamenjavo koordinatnega sistema [AS76].

Integracija po delih

Pomembna lastnost formulacije z uteženim ostankom je, da lahko v primeru odvedljivih testnih funkcij na integralu v enačbi 2.3 uporabimo pravilo za integracijo *po delih* (lat. per partes) [Hea02]:

$$\int_a^b f'g dx = fg|_a^b - \int_a^b fg' dx.$$

V več prostorskih dimenzijah sorodno pravilo izpeljemo iz *izreka o divergenci* [War83]: naj bo $\Omega \subset R^n$ gladka kompaktna mnogoterost dimenzije n. Njen rob Γ je torej gladka sklenjena mnogoterost dimenzije $n-1$, ki ima lahko vogale. Naj bo $\mathbf{F}(\mathbf{x}) = [f_i(\mathbf{x})]$ gladka vektorska funkcija, definirana na Ω. Potem velja:

$$\int_\Omega \nabla \cdot \mathbf{F}\, d\Omega = \int_\Gamma \mathbf{F} \cdot \mathbf{n}\, d\Gamma, \tag{2.5}$$

kjer je $\nabla \cdot \mathbf{F} = \sum_{i=1}^n \frac{\partial f_i}{\partial x_i}$ *divergenca* vektorske funkcije $\mathbf{F}$, $\mathbf{n}$ pa *zunanji normalni vektor* Γ. Vzemimo zdaj vektorsko funkcijo, katere $i-$ta komponenta je zmnožek funkcij f in g, ostale komponente pa so enake 0: $\mathbf{F}(\mathbf{x}) = [0,\ldots,0,f(\mathbf{x})g(\mathbf{x}),0,\ldots,0]$. Potem velja:

$$\nabla \cdot \mathbf{F} = \frac{\partial f}{\partial x_i} g + f \frac{\partial g}{\partial x_i}, \qquad \mathbf{F} \cdot \mathbf{n} = fg\, n_i,$$

kjer je n_i komponenta $\mathbf{n}$ v smeri x_i. Z vstavljanjem obeh izrazov v (2.5) in prenašanjem enega od členov z leve na desno dobimo končno obliko "integracije po delih" v več dimenzijah:

$$\int_\Omega \frac{\partial f}{\partial x_i} g \, d\Omega = \int_\Gamma f g \, n_i \, d\Gamma - \int_\Omega f \frac{\partial g}{\partial x_i} \, d\Omega.$$

Če na primer v PDE nastopa člen $\frac{\partial^r u}{\partial x_i^r}$, pripadajoči člen integrala (2.3) pretvorimo:

$$\int_\Omega W \frac{\partial^r u}{\partial x_i^r} d\Omega = \int_\Gamma W \frac{\partial^{r-1} u}{\partial x_i^{r-1}} d\Gamma - \int_\Omega \frac{\partial W}{\partial x_i} \frac{\partial^{r-1} u}{\partial x_i^{r-1}} d\Omega. \tag{2.6}$$

Na ta način pretvorimo izbrane člene in dobljeno alternativno obliko enačbe 2.3 uporabimo za tvorbo sistema navadnih enačb na enak način, kot smo dobili (2.4). Bazne funkcije $\mathbf{\Phi}$ morajo biti v tem primeru le $r-1$-krat namesto r-krat odvedljive.

2.2 Difuzijska enačba

Za študijski primer bomo vzeli *linearno difuzijsko enačbo* [CJ59, Ö94]:

$$\frac{\partial u}{\partial t} - c\nabla^2 u - g = 0, \tag{2.7}$$

kjer je $u(\mathbf{x}, t)$ iskana funkcija časa in prostorskih koordinat, c je *difuzijska konstanta*, $g(\mathbf{x}, t)$ pa *funkcija izvorov* (angl. source function). Operator ∇ (*nabla*) je vektorski operator parcialnega odvajanja:

$$\nabla = \frac{\partial}{\partial x_i} = \left(\frac{\partial}{\partial x_1}, \cdots, \frac{\partial}{\partial x_n} \right), \qquad \nabla^2 = \sum_i \frac{\partial^2}{\partial x_i^2} \, .$$

Potrebujemo še opis domene Ω, na kateri bi enačbo radi rešili, roba Γ domene ter začetne in robne pogoje:

$$u(\mathbf{x}, 0), \ \mathbf{x} \in \Omega; \qquad u(\mathbf{x}, t), \ \mathbf{x} \in \Gamma, t > 0.$$

Difuzijsko enačbo smo izbrali zaradi splošnosti, saj vsebuje tudi časovno odvisnost, stabilnosti (v primeru $c > 0$), ki olajša formulacijo metod in tako omogoči, da se bolj posvetimo njihovi časovni zahtevnosti in paralelizaciji, in zaradi uporabnosti. Opisuje namreč različne naravne pojave, na primer prenos toplote v homogeni trdni snovi, mešanje snovi z različnimi koncentracijami in konsolidacijo tal v gradbeništvu [CJ59].

V primeru prenosa toplote rešitev u predstavlja temperaturo, $c = \frac{\lambda}{\rho c_p}$ je skupek snovnih konstant (koeficient toplotne prevodnosti λ, gostota ρ in specifična toplota c_p), g pa so toplotni izvori in ponori. Kvalitativno lahko enačbo razložimo prek vplivov zadnjih dveh členov na časovni odvod temperature. Člen $c\nabla^2 u$ opisuje prevajanje toplote: če je drugi prostorski odvod u v neki točki pozitiven, je ta točka hladnejša od povprečne temperature

njene najožje okolice, zato se ji temperatura povečuje (člen pozitivno vpliva na časovni odvod temperature). Prevajanje toplote je hitrejše skozi bolj prevodne snovi (večji λ), počasnejše pa skozi toplotno gostejše snovi (večji produkt ρc_p). Pozitivna vrednost člena g opisuje izvore toplote – temperatura se bo na tem mestu povečevala – in nasprotno pri negativni vrednosti.

2.2.1 Enodimenzijska in dvodimenzijska oblika

Dasiravno so naravni pojavi tridimenzionalni, bomo v tem delu obravnavali dvodimenzijsko obliko enačbe 2.7:

$$\frac{\partial u(x,y,t)}{\partial t} - c\left[\frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2}\right] - g = 0, \tag{2.8}$$

oziroma krajše:

$$u_t - c(u_{xx} + u_{yy}) - g = 0.$$

Pri funkciji izvorov g bomo z namenom kompaktnejšega zapisa izpuščali odvisnost od koordinat in časa.

Dvodimenzijska difuzijska enačba je od tridimenzijske geometrijsko enostavnejša in omogoča lažjo vizualizacijo metod in rezultatov. Konceptualno sta ti dve obliki enakovredni – razširitev na tri dimenzije zgolj napihne število členov v izpeljavah in število vejitev v programu, ki obravnava različne odnose med diskretnimi deli domene. Zavedati pa se moramo, da za splošne dvodimenzionalne domene obstajajo povsem avtomatični postopki izdelave mreže končnih elementov, za tridimenzionalne pa ne [IOCP03].

Enodimenzijska oblika:

$$\frac{\partial u(x,t)}{\partial t} - c\frac{\partial^2 u(x,t)}{\partial x^2} - g = 0 \tag{2.9}$$

je konceptualno enostavnejša, zato zabriše razlike med metodami z enostavno in zapleteno formulacijo. Vseeno je včasih koristna v ilustrativne namene.

2.2.2 Analitična rešitev

Difuzijsko enačbo lahko analitično rešimo le v posebnih primerih. Poleg trivialnih, recimo s konstantnimi začetnimi in robnimi pogoji ter brez izvorov in ponorov:

$$u(\mathbf{x}, 0) = u(\Gamma, t) = a,\ g = 0,$$

je znana rešitev na enotskem kvadratu $\Omega = [0,1] \times [0,1]$ z robnimi in začetnimi pogoji:

$$u(0,y,t) = u(1,y,t) = u(x,0,t) = u(x,1,t) = 0,$$

$$u(x,y,0) = 1,\ x,y \neq 0,1$$

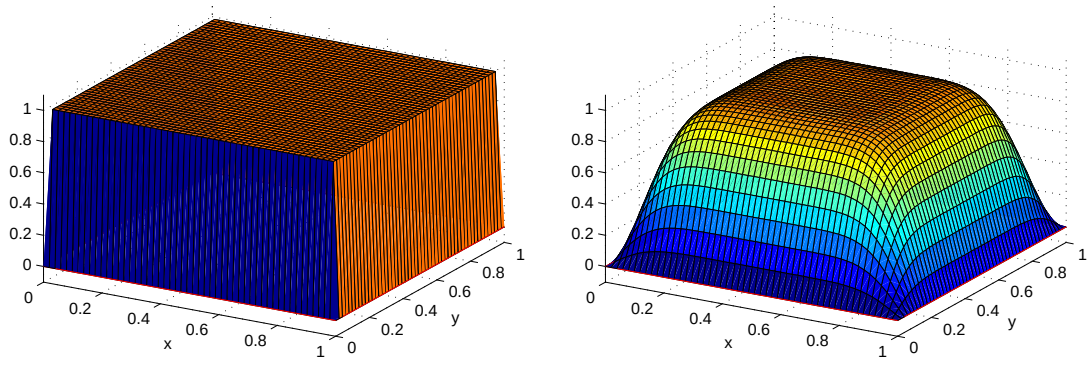
ter brez izvorov in ponorov [CJ59]. Rešitev je vsota vrste:

$$u(x,y,t) = \sum_{n=0}^{\infty}\sum_{m=0}^{\infty} L_{n,m}\cos((2n+1)\pi(x-0.5))\cos((2m+1)\pi(y-0.5))e^{-t\cdot D_{n,m}}, \quad (2.10)$$

kjer je

$$L_{n,m} = \frac{16}{\pi^2}\cdot\frac{(-1)^{n+m}}{(2n+1)(2m+1)},$$
$$D_{n,m} = c\pi^2\left[(2n+1)^2+(2m+1)^2\right].$$

Za takšen preprost primer torej znamo poljubno natančno izračunati vrednost točne rešitve in s tem napako približne rešitve, dobljene z neko numerično metodo. Slika 2.1 prikazuje opisane začetne pogoje in rešitev za primer $c = 0.001$ ob času $t = 5$ v brezdimenzijskih enotah, kar ustreza neskončno dolgi aluminijasti palici s kvadratnim presekom 1m × 1m z začetno temperaturo 1°C in robno 0°C po času 51.1 s. Konstante aluminija so namreč $\lambda = 235\frac{\mathrm{W}}{\mathrm{mK}}$, $\rho = 2700\frac{\mathrm{kg}}{\mathrm{m}^3}$, $c_p = 890\frac{\mathrm{J}}{\mathrm{kgK}}$ [ALU] in $c = \frac{\lambda}{\rho c_p} = 9.78\cdot 10^{-5}$.



Slika 2.1: Začetni pogoji za dvodimenzijsko difuzijsko enačbo na enotskem kvadratu (levo) in rešitev ob času $t = 5$ (desno).

2.3 Časovna diskretizacija

Kot smo že povedali, s prostorsko diskretizacijo časovno odvisne PDE dobimo sistem NDE, ki ga v sistem navadnih enačb pretvorimo s časovno diskretizacijo. Teoretično bi lahko čas obravnavali enako kot ostale neodvisne spremenljivke, vendar se ga v praksi loči, ker je njegov fizikalni pomen drugačen. Poleg tega je domena v prostoru zaprta z vseh strani, čas pa je običajno odprt v eno smer.

V tem delu se bomo omejili na PDE, ki so v času prvega reda, kot je na primer difuzijska enačba. V nasprotnem bi lahko dobljeni sistem NDE pretvorili v večji sistem prvega reda,

kot smo pokazali v uvodu. Tako bo imel dobljeni sistem NDE vedno eksplicitno obliko:

$$\mathbf{u}' = \mathbf{f}(t, \mathbf{u}), \tag{2.11}$$

kjer so iskane funkcije $u_i(t)$ vrednosti $\hat{u}$ v vozliščih ali parametri poskusne funkcije, uporabljene za prostorsko diskretizacijo. Podani so tudi začetni pogoji $\mathbf{u}(0)$, ki določajo, katera partikularna rešitev iz družine splošnih rešitev enačbe 2.11 je prava. Zadovoljili se bomo s približkom rešitve v diskretnih trenutkih, ki ga bomo označili z

$$\mathbf{u}^{(i)} = \mathbf{u}(t_i),\ t_i = i\Delta t,$$

kjer je Δt *časovni korak*.

2.3.1 Eulerjeva metoda in red metode

Zapišimo Taylorjev razvoj točne rešitve sistema (2.11) okrog točke t_i:

$$\mathbf{u}(t_i + \Delta t) = \mathbf{u}(t_i) + \mathbf{u}'(t_i)\Delta t + \frac{\mathbf{u}''(\xi)}{2}(\Delta t)^2;\quad \xi \in (t_i, t_i + \Delta t).$$

Če zanemarimo člen z $(\Delta t)^2$, dobimo *eksplicitno Eulerjevo metodo* [Ore97]:

$$\mathbf{u}(t_{i+1}) = \mathbf{u}(t_i) + \Delta t\mathbf{f}(t_i, \mathbf{u}(t_i)), \tag{2.12}$$

s katero po vrsti izračunamo vse vrednosti $\mathbf{u}(t_i)$ in s tem rešimo enačbo. V bistvu sistema NDE nismo pretvorili v sistem navadnih enačb, marveč smo dobili kar postopek za izračun vrednosti njegove rešitve, oziroma smo dobili trivialen sistem.

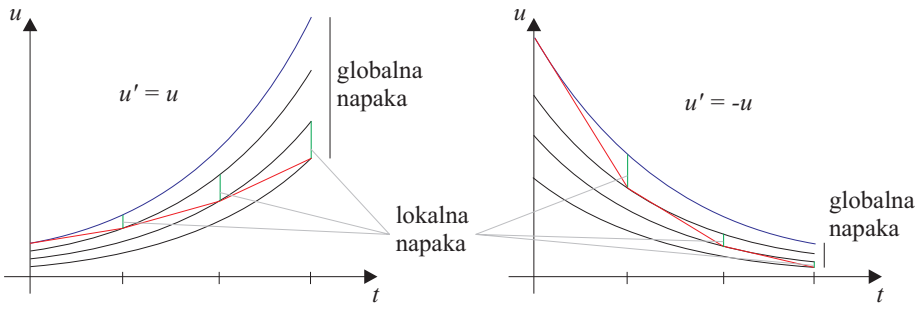
Ker smo upoštevali člene Taylorjeve vrste do vključno tistega, ki je sorazmeren Δt, pravimo, da je Eulerjeva metoda *prvega reda* [Ore97]. Napaka v posameznem koraku je sorazmerna prvemu zanemarjenemu členu, torej $(\Delta t)^2$. Metoda reda k bi imela napako v vsakem koraku sorazmerno $(\Delta t)^{k+1}$ in vsoto napak posameznih korakov sorazmerno $(\Delta t)^k$, ker je število korakov sorazmerno $1/\Delta t$.

2.3.2 Stabilnost in natančnost

Napaka v posameznem koraku je *lokalna napaka* L_i, to je razlika med numeričnim približkom $\mathbf{u}^{(i)}$ in tisto rešitvijo diferencialne enačbe, ki poteka skozi predhodno točko $(t_{i-1}, \mathbf{u}^{(i-1)})$. *Globalna napaka* G_i pa je razlika med numeričnim približkom $\mathbf{u}^{(i)}$ in iskano rešitvijo enačbe, ki poteka skozi točko $(0, \mathbf{u}(0))$. Slika 2.2 prikazuje odnos med lokalno in globalno napako na primeru enačb $u' = u$ in $u' = -u$ [Hea02].

Zveza med lokalno in globalno napako je odvisna od enačbe. Definirajmo *Jacobijevo matriko*, katere elementi so:

$$J_{ij}(t) = \left.\frac{\partial f_i}{\partial u_j}\right|_t.$$



Slika 2.2: Odnos med lokalno in globalno napako. Iskana rešitev je prikazana modro, ostale partikularne rešitve črno, z eksplicitno Eulerjevo metodo dobljeni približek rdeče in lokalne napake zeleno.

Če so realni deli vseh lastnih vrednosti matrike J negativni, se z naraščanjem t partikularne rešitve približujejo in je globalna napaka manjša od vsote lokalnih napak – takšne NDE imenujemo *stabilne*. Primer je enačba $u' = -u$, katere Jacobijeva matrika je skalar $J = -1$. *Nestabilne* so enačbe, pri katerih obstaja lastna vrednost λ matrike J z $\mathrm{Re}(\lambda) \geq 0$, na primer $u' = u, J = 1$. Njihove partikularne rešitve se oddaljujejo in so težavnejše za numerično reševanje, saj je globalna napaka večja od vsote lokalnih [Hea02]. Opozoriti je treba, da sta Jacobijeva matrika in z njo stabilnost v splošnem odvisni od časa t in izbrane partikularne rešitve.

V uvodu smo definirali *stabilnost numerične metode* kot neobčutljivost rezultata na majhne spremembe začetnih podatkov. Stabilnost metode lahko preverimo na primeru NDE $u' = \lambda u$ z začetnim pogojem $u(0) = u_0$, katere točna rešitev je $u(t) = u_0 e^{\lambda t}$. Rešitev se asimptotično približuje ničli, če je $\lambda < 0$. Eksplicitna Eulerjeva metoda bo vrnila rešitev:

$$u(t_i) = u(t_{i-1}) + \Delta t \lambda u(t_{i-1}) = (1 + \Delta t \lambda) u(t_{i-1}) = (1 + \Delta t \lambda)^i u_0.$$

Približna rešitev za to PDE se bliža 0, če je absolutna vrednost *faktorja ojačitve* $(1+\Delta t\lambda)$ manjša od 1: $|1 + \Delta t\lambda| < 1$. V primeru enačbe $u' = -u$ je torej pogoj za stabilnost eksplicitne Eulerjeve metode $\Delta t < 2$. Splošni pogoj za stabilnost dobimo, če namesto λ vstavimo J. V tem primeru mora veljati $\rho(I + \Delta t J) < 1$, kjer je I enotska matrika ustrezne velikosti in ρ *spektralni radij*, to je največja absolutna vrednost lastnih vrednosti matrike [Hea02].

2.3.3 Implicitne metode

Eulerjeva metoda je *eksplicitna*, saj je v enačbi 2.12 nova vrednost $\mathbf{u}^{(i+1)}$ izražena z že znanimi vrednostmi. Drugače je pri *implicitni* Eulerjevi metodi [Hea02]:

$$\mathbf{u}(t_{i+1}) = \mathbf{u}(t_i) + \Delta t\, \mathbf{f}(t_{i+1}, \mathbf{u}(t_{i+1})), \tag{2.13}$$

kjer nastopa vrednost $\mathbf{f}$ v še neznani točki. Dobili smo sistem navadnih enačb, ki ga moramo rešiti v vsakem koraku in katerega zapletenost je odvisna od funkcije $\mathbf{f}$.

Iz NDE $u' = \lambda u$ dobimo enačbo:

$$u(t_{i+1}) = u(t_i) + \Delta t \lambda u(t_{i+1}), \qquad \text{oziroma} \qquad u(t_{i+1}) = \frac{1}{1 - \Delta t\lambda} u(t_i).$$

Faktor ojačitve je $\frac{1}{1-\Delta t\lambda}$, kar je v primeru stabilne enačbe ($\lambda < 0$) manj kot 1 za katerikoli pozitivni Δt, zato je metoda stabilna ne glede na velikost koraka oziroma *A-stabilna*. Tudi za vse druge (stabilne) DE velja, da je implicitna Eulerjeva metoda stabilna ne glede na velikost koraka, zato moramo pri izbiri Δt upoštevati le želeno natančnost. Tudi ostale implicitne metode, ne le Eulerjeva, imajo v splošnem večje intervale stabilnosti od eksplicitnih, niso pa vse A-stabilne [Hea02].

Natančnost implicitne Eulerjeve metode lahko preverimo s primerjavo faktorja ojačitve in Taylorjevega razvoja točne rešitve NDE $u' = \lambda u$ z začetnim pogojem $u(0) = 1$ okrog točke 0:

$$u(\Delta t) = e^{\Delta t\lambda} = 1 + \Delta t\lambda + \frac{(\Delta t\lambda)^2}{2} + \frac{(\Delta t\lambda)^3}{6} + O((\Delta t)^4).$$

Lokalno napako dobimo s preverjanjem enakosti:

$$\begin{aligned} \frac{1}{1 - \Delta t\lambda} &\approx 1 + \Delta t\lambda + \frac{(\Delta t\lambda)^2}{2} + O((\Delta t)^3), \\ 1 &\approx 1 - \Delta t\lambda + \Delta t\lambda - (\Delta t\lambda)^2 + \frac{(\Delta t\lambda)^2}{2} + O((\Delta t)^3), \\ 0 &\approx -\frac{(\Delta t\lambda)^2}{2} + O((\Delta t)^3). \end{aligned}$$

Lokalna napaka je sorazmerna $(\Delta t)^2$, torej je metoda enako kot eksplicitna Eulerjeva metoda prvega reda natančnosti.

Natančnejša *Crank-Nicolsonova* shema [Fle88] definira odvod rešitve v vmesni točki $t + \frac{1}{2}\Delta t$ kot:

$$\mathbf{u}'(t + \frac{1}{2}\Delta t) = \frac{\mathbf{u}(t + \Delta t) - \mathbf{u}(t)}{\Delta t}$$

in rešitev v isti točki z linearno interpolacijo naslednje in prejšnje vrednosti:

$$\mathbf{u}(t + \frac{1}{2}\Delta t) = \frac{\mathbf{u}(t) + \mathbf{u}(t + \Delta t)}{2}.$$

Ko vstavimo izraza v (2.11), dobimo:

$$\frac{\mathbf{u}(t + \Delta t) - \mathbf{u}(t)}{\Delta t} = \mathbf{f}\left(t + \frac{1}{2}\Delta t, \frac{\mathbf{u}(t) + \mathbf{u}(t + \Delta t)}{2}\right),$$

oziroma:

$$\mathbf{u}(t + \Delta t) = \mathbf{u}(t) + \Delta t\, \mathbf{f}\left(t + \frac{1}{2}\Delta t, \frac{\mathbf{u}(t) + \mathbf{u}(t + \Delta t)}{2}\right).$$

Vstavimo v shemo enačbo $u' = \lambda u$:

$$u(t_{i+1}) = u(t_i) + \Delta t\lambda \frac{u(t_i) + u(t_{i+1})}{2},$$
$$u(t_{i+1}) = \frac{2 + \Delta t\lambda}{2 - \Delta t\lambda} u(t_i).$$

Tudi ta shema je A-stabilna, ker je pri $\lambda < 0$ faktor ojačitve med -1 in 1 za vsak pozitivni Δt. Na enak način kot pri implicitni Eulerjevi metodi preverimo tudi natančnost Crank-Nicolsonove sheme:

$$\frac{2 + \Delta t\lambda}{2 - \Delta t\lambda} \approx 1 + \Delta t\lambda + \frac{(\Delta t\lambda)^2}{2} + \frac{(\Delta t\lambda)^3}{6} + O((\Delta t)^4),$$
$$2 + \Delta t\lambda \approx 2 - \Delta t\lambda + 2\Delta t\lambda - (\Delta t\lambda)^2 + (\Delta t\lambda)^2 - \frac{(\Delta t\lambda)^3}{2} + \frac{(\Delta t\lambda)^3}{3} + O((\Delta t)^4),$$
$$0 \approx -\frac{(\Delta t\lambda)^3}{6} + O((\Delta t)^4).$$

Lokalna napaka je torej sorazmerna $(\Delta t)^3$, metoda je drugega reda natančnosti. Dobljeni sistem navadnih enačb je podobne težavnosti kot pri implicitni Eulerjevi metodi. Če je $\mathbf{f}$ linearna, lahko Crank-Nicolsonovo shemo izpeljemo tudi kot povprečje eksplicitne in implicitne Eulerjeve metode [Hea02].

2.4 Metoda končnih razlik

Metoda končnih razlik (MKR, angl. finite difference method (FDM)) aproksimira rešitev z vrednostmi v točkah pravilne, običajno ekvidistantne mreže, v katerih navadne enačbe tvori z uporabo kolokacije [Hea02]. V dveh dimenzijah, diskretiziranih z enako ločljivostjo, je približek rešitve množica vrednosti:

$$u_{i,j} = \hat{u}(i\Delta x, j\Delta x),\ (i\Delta x, j\Delta x) \in \Omega.$$

V *notranjih točkah*, katerih vse štiri sosede so vsebovane v Ω, izrazimo ostanek, pri čemer za aproksimacijo odvodov uporabimo formuli (2.2). Za difuzijsko enačbo dobimo sistem NDE [Ö94, PŠT04]:

$$u'_{i,j} - c\left[\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{(\Delta x)^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{(\Delta x)^2}\right] - g = 0.$$

Vrednosti u v robnih točkah izenačimo s predpisanimi robnimi pogoji. Natančnost metode je istega reda kot uporabljeni približki odvodov, torej bo napaka sorazmerna $(\Delta x)^2$.

Čas diskretiziramo z enakomernimi koraki Δt in označimo $u_{i,j}^{(k)} = u_{i,j}(k\Delta t)$. Eksplicitna Eulerjeva metoda nam da postopek izračuna novih vrednosti iz starih:

$$u_{i,j}^{(k+1)} = u_{i,j}^{(k)} + \Delta t\left[c\frac{u_{i-1,j}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i,j+1}^{(k)} - 4u_{i,j}^{(k)}}{(\Delta x)^2} + g\right].$$

Mnogo boljša je Crank-Nicolsonova shema, vendar v vsakem časovnem koraku zahteva reševanje sistema enačb. V obravnavanem primeru jo najlažje izpeljemo kot povprečje implicitne in eksplicitne Eulerjeve metode:

$$u_{i,j}^{(k+1)} = u_{i,j}^{(k)} + \Delta t \cdot c \cdot \frac{1}{2} \left[\frac{u_{i-1,j}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i,j+1}^{(k)} - 4u_{i,j}^{(k)}}{(\Delta x)^2} + \frac{u_{i-1,j}^{(k+1)} + u_{i+1,j}^{(k+1)} + u_{i,j-1}^{(k+1)} + u_{i,j+1}^{(k+1)} - 4u_{i,j}^{(k+1)}}{(\Delta x)^2} \right] + \Delta t \cdot g.$$

Če vse vrednosti $u_{i,j}$ zberemo v vektor $\mathbf{u}$, lahko sistem lahko preuredimo v običajnejšo obliko $A\mathbf{u}^{(k+1)} = B\mathbf{u}^{(k)} + \mathbf{f}$ oziroma:

$$(2I + \Delta t \cdot K)\mathbf{u}^{(k+1)} = (2I - \Delta t \cdot K)\mathbf{u}^{(k)} + 2\Delta t\, \mathbf{g}, \tag{2.14}$$

kjer je I enotska matrika ustrezne velikosti in

$$K_{p,q} = \frac{1}{(\Delta x)^2} \cdot \begin{cases} 4c & p = q \\ -c & p \text{ in } q \text{ sta sosednji točki} \\ 0 & \text{sicer,} \end{cases}$$

v vektorju $\mathbf{g}$ pa so zbrane vrednosti g v času $(k + \frac{1}{2})\Delta t$. V vsakem časovnem koraku moramo torej opraviti eno množenje matrika-vektor, eno seštevanje vektorjev in rešiti sistem. Matriki A in B sta *redki*, njuna struktura pa je odvisna od načina oštevilčenja točk. Če pri oštevilčevanju izpustimo robne točke, s čimer se njihov vpliv preseli v $\hat{\mathbf{f}} = \mathbf{f} +$ vpliv robnih točk, bosta matriki tudi *simetrični*, kar pride prav pri reševanju sistema.

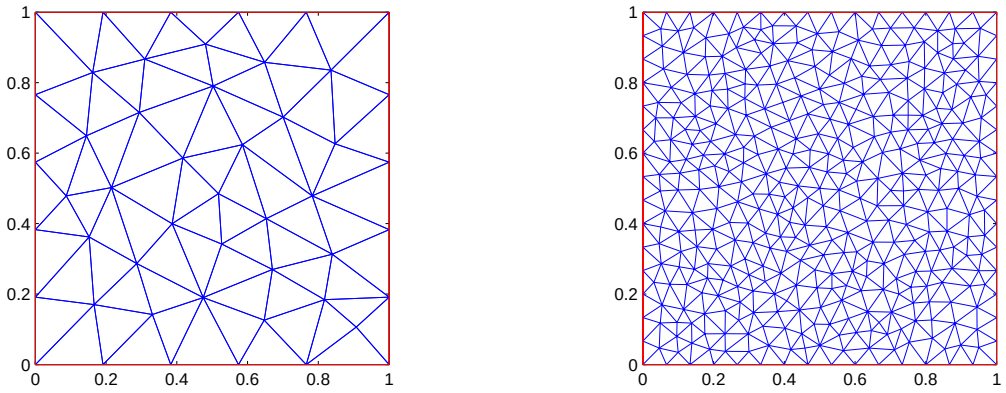
2.5 Metoda končnih elementov

Metoda končnih elementov (MKE, angl. finite element method (FEM)), ki skupaj s pripadajočo terminologijo izhaja iz prvotne uporabe v strojništvu, domeno razdeli na enostavne *elemente* [Kat03, Buc95]. To so liki, ki zapolnijo prostor, na primer daljice v eni dimenziji, nepravilni trikotniki ali četverokotniki v dveh in nepravilni tetraedri ali heksaedri v treh dimenzijah. Vozlišča so oglišča likov, ki morajo biti postavljeni tako, da se vedno stikajo bodisi s celo stranico ali pa le z ogliščem, ne pa na primer s polovico stranice. Vsako vozlišče je tako oglišče vseh elementov, ki se jih dotika. Paziti je treba tudi, da se v posameznem vozlišču ne stika preveliko število elementov in da elementi niso izrojeni, na primer zelo sploščeni.

MKE lahko reši PDE tako, da približek (poskusno funkcijo $\hat{u}$) in ostanek predstavi z linearno kombinacijo baznih funkcij ter sestavi enačbe za minimizacijo ostanka bodisi s kolokacijo ali Galerkinovo metodo [Hea02], ki jo uporabljamo v tem delu. Obstajajo tudi druge različice, zasnovane na primer na *variacijskem načelu* (angl. variational principle) [Buc95].

2.5.1 Izdelava mreže

Za izdelovanje mreže obstaja množica postopkov, ki zagotovijo, da dobljena mreža ustreza nekemu kriteriju, povezanemu s kakovostjo elementov [Owe98]. Najpogosteje se uporablja *Delaunayev kriterij* [Del34]: če skozi tri vozlišča, ki tvorijo trikotni element, potegnemo krožnico, ne sme znotraj nje ležati nobeno drugo vozlišče. Dva primera razdelitve enotskega kvadrata na trikotnike z upoštevanjem Delaunayevega kriterija prikazuje slika 2.3.



Slika 2.3: Razdelitev enotskega kvadrata na 76 in 740 trikotnikov.

V primeru treh prostorskih dimenzij se kriterij glasi: če skozi štiri vozlišča, ki tvorijo element oblike nepravilnega tetraedra, potegnemo plašč krogle, ne sme znotraj nje ležati nobeno drugo vozlišče. Na hude težave naletimo, ko več kot štiri vozlišča ležijo na plašču iste krogle. Na Delaunayevem kriteriju zasnovani postopki v takih primerih dopuščajo možnost sploščenih elementov (angl. slivers), odprava katerih največkrat zahteva človeško intervencijo in premikanje vozlišč [IOCP03]. Zaradi opisane in podobnih težav problem avtomatičnega generiranja mreže elementov v treh dimenzijah še ni dokončno rešen [FM04, Liu03].

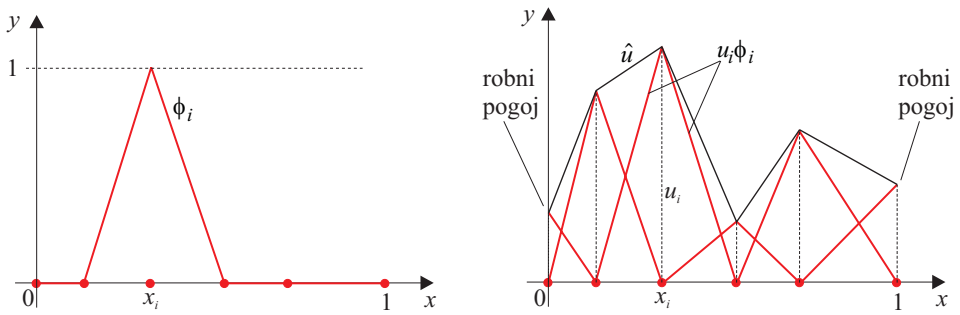
2.5.2 Bazne funkcije

Poskusno funkcijo $\hat{u}$ sestavimo iz baznih funkcij ϕ z lokalnim nosilcem, ki imajo vrhove v vozliščih in v vseh straneh padajo proti 0. Zanje velja *Kroneckerjeva δ lastnost*, da je vrednost funkcije 1 v enem izmed vozlišč in 0 v vseh ostalih vozliščih:

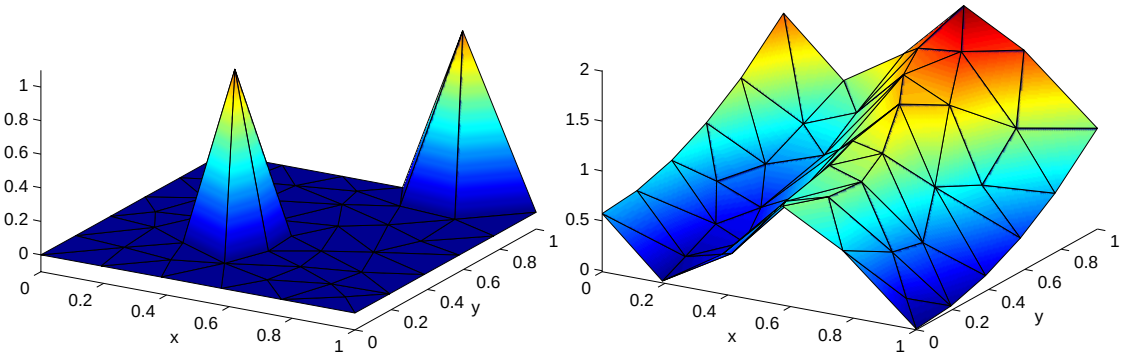
$$\phi_i(\mathbf{x}_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases} \tag{2.15}$$

Bazne funkcije prve stopnje so linearne, če so elementi daljice, trikotniki ali nepravilni tetraedri, in bilinearne ali trilinearne, če so elementi pravokotniki ali nepravilni heksa-

edri. Segajo natanko en element daleč, kot vidimo na sliki 2.4. Funkcije višjih redov lahko segajo dlje, kar pomeni, da morajo zaradi Kroneckerjeve δ lastnosti imeti prenihaje [Hea02]. Na sliki 2.5 je še dvodimenzionalen primer.



Slika 2.4: Linearna bazna funkcija v eni dimenziji (levo). Bazne funkcije, pomnožene z vozliščnimi parametri (črtkano) sestavljajo poskusno funkcijo $\hat{u}$ (črno).



Slika 2.5: Dve linearni bazni funkciji v dveh dimenzijah (levo) in primer dvodimenzionalne poskusne funkcije (desno).

Koeficientu u_i linearne kombinacije pravimo *parameter vozlišča* $\mathbf{x}_i$ oziroma *vozliščni parameter* (angl. nodal parameter). Pomembna posledica Kroneckerjeve δ lastnosti je, da je vrednost poskusne funkcije v vsakem vozlišču enaka pripadajočemu parametru:

$$\hat{u}(\mathbf{x}_i) = \sum_j u_j \phi_j(\mathbf{x}_i) = u_i,$$

kar nam omogoča enostavno vsiljevanje Dirichletovih robnih pogojev – parameter u_i kar izenačimo z vrednostjo, predpisano v robnem vozlišču $\mathbf{x}_i$. Kot bomo videli pozneje, nekatere različice mreže prostih metod te lastnosti nimajo, zato je vsiljevanje robnih pogojev težje [Liu03].

2.5.3 Metoda Galerkina

Metoda Galerkina ustvari sistem enačb, ki zahtevajo, da je ostanek ortogonalen baznim funkcijam. Za i-to notranje vozlišče dobimo enačbo:

$$\int_{\Omega_{Qi}} r(\mathbf{x}, \mathbf{u})\phi_i(\mathbf{x})d\Omega = 0. \tag{2.16}$$

Namesto po celotni domeni Ω lahko integriramo le po nosilcu funkcije ϕ_i, saj je drugje ϕ_i enaka 0. Okolico vozlišča, po kateri integriramo (2.16), bomo imenovali *integracijska* ali *kvadraturna domena* vozlišča $\mathbf{x}_i$ in označevali z Ω_{Qi}.

Za primer rešimo enačbo 2.1 s strani 16, ki smo jo v razdelku 2.1.1 reševali s kolokacijo. Spet vzamemo eno notranje vozlišče $x_1 = \frac{1}{2}$ z bazno funkcijo, podobno tisti na sliki 2.4:

$$\phi_1(x) = \begin{cases} 2x & x \leq 0.5 \\ 2 - 2x & x > 0.5, \end{cases} \qquad \phi_1'(x) = \begin{cases} 2 & x < 0.5 \\ -2 & x > 0.5. \end{cases}$$

Imamo torej dva elementa – daljici $[0, x_1]$ in $[x_1, 1]$. V vsakem izmed elementov sta bazna funkcija ϕ_1 in njen odvod zvezna. Oba robna pogoja sta 0, zato morata biti tudi parametra robnih vozlišč 0 in posledično njuni bazni funkciji nista pomembni. Izrazimo ostanek z edino neznanko u_1:

$$r(x, u_1) = \hat{u}''(x) - 12x^2 = u_1\phi_1''(x) - 12x^2.$$

Bazna funkcija ϕ_1 ni dvakrat zvezno odvedljiva, saj ima že prvi odvod nezveznost pri $x = 0.5$, zato ta zapis matematično postane smiseln šele, ko na dobljenih integralih uporabimo integracijo po delih. Edina enačba dobljenega linearnega sistema se glasi:

$$\int_0^1 \left[u_1\phi_1''(x) - 12x^2\right]\phi_1(x)dx = 0$$

in se z integracijo po delih ter množenjem cele enačbe z -1 spremeni v:

$$-\left[u_1\phi_1'(x)\phi_1(x)\right]_0^1 + \int_0^1 u_1\phi_1'(x)\phi_1'(x)dx + \int_0^1 12x^2\phi_1(x)dx = 0.$$

Prvi člen odpade, ker je ϕ_1 na robovih enaka 0. Integral lahko rešimo analitično po elementih in dobimo končno enačbo:

$$\begin{aligned} [u_1 \cdot 4x]_0^{\frac{1}{2}} + [u_1 \cdot 4x]_{\frac{1}{2}}^1 + [6x^4]_0^{\frac{1}{2}} + [8x^3 - 6x^4]_{\frac{1}{2}}^1 &= 0, \\ 2u_1 + 4u_1 - 2u_1 + \frac{3}{8} + 8 - 6 - 1 + \frac{3}{8} &= 0, \\ 4u_1 &= -\frac{14}{8} \end{aligned}$$

ter rešitev $u_1 = \hat{u}(\frac{1}{2}) = -\frac{7}{16}$, ki je enaka točni rešitvi. V splošnem primeru MKE ni točna, vendar je natančnejša od MKR.

Iz zgodovinskih razlogov se matrika sistema (v našem primeru skalar -4) pogosto označuje s K in imenuje *togostna matrika* (angl. stiffness matrix), vektor desnih strani (v našem primeru skalar 7/16) pa s $\mathbf{f}$ in imenuje *vektor sil* (angl. load vector) [Buc95].

2.5.4 Lokalno obravnavanje elementov

V bolj realnem primeru z mnogo elementi lahko zgornjo izpeljavo naredimo za en splošen element, kar nam da *lokalno togostno matriko* K^L. Ta je velikosti $n \times n$, kjer je n število vozlišč na element (na primer 3 za trikotnike), $K^L_{k,l}$ pa opisuje lokalni odnos med vozliščema k in l. Globalno matriko K dobimo iz lokalnih prispevkov po naslednjem postopku:

1. $K = 0$.
2. Iz seznama elementov izberi enega in izračunaj K^L.
3. Za vsak element lokalne matrike $K^L_{k,l}$: če ima k-to vozlišče izbranega elementa indeks i v globalni tabeli vozlišč, l-to vozlišče pa indeks j, povečaj $K_{i,j}$ za $K^L_{k,l}$.
4. Ponavljaj koraka 2 in 3, dokler ne obdelaš vseh elementov.

Podobno lahko tudi vektor sil sestavimo iz lokalnih prispevkov.

2.5.5 Reševanje dvodimenzijske difuzijske enačbe

Difuzijsko enačbo

$$u_t - c(u_{xx} + u_{yy}) - g = 0 \tag{2.17}$$

rešimo na podoben način kot primer iz razdelka 2.5.3. Definirajmo poskusno funkcijo:

$$\hat{u}(x,y,t) = \sum_{j=1}^{N} u_j(t)\phi_j(x,y), \qquad \hat{u}_t = \sum_{j=1}^{N} \frac{\partial u_j(t)}{\partial t}\phi_j(x,y) = \sum_{j=1}^{N} u'_j\phi_j,$$

$$\hat{u}_x = \sum_{j=1}^{N} u_j(t)\frac{\partial \phi_j(x,y)}{\partial x} = \sum_{j=1}^{N} u_j\phi_{j,x}, \qquad \hat{u}_{xx} = \sum_{j=1}^{N} u_j(t)\frac{\partial^2 \phi_j(x,y)}{\partial x^2} = \sum_{j=1}^{N} u_j\phi_{j,xx},$$

kjer je N celotno število vozlišč v domeni. Podobno izrazimo tudi odvode po y.

Galerkinove enačbe zahtevajo, da je ostanek, to je leva stran (2.17), ortogonalen baznim funkcijam:

$$\int_\Omega \phi_i \sum_{j=1}^{N} u'_j\phi_j d\Omega - c\int_\Omega \phi_i \sum_{j=1}^{N} u_j \left[\phi_{j,xx} + \phi_{j,yy}\right] d\Omega - \int_\Omega \phi_i g d\Omega = 0.$$

Ena takšna enačba pripada vsakemu notranjemu vozlišču $\mathbf{x}_i$. Najprej premaknemo operatorje seštevanja in vozliščne parametre pred integrale:

$$\sum_{j=1}^{N} u'_j \int_\Omega \phi_i\phi_j d\Omega - c\sum_{j=1}^{N} u_j \int_\Omega \left[\phi_i\phi_{j,xx} + \phi_i\phi_{j,yy}\right] d\Omega - \int_\Omega \phi_i g d\Omega = 0. \tag{2.18}$$

Drugih odvodov baznih funkcij $\phi_{j,xx}$ in $\phi_{j,yy}$ se znebimo z uporabo oblike izreka o divergenci (2.6), ki ga pripravimo za uporabo na drugem integralu enačbe 2.18:

$$\int_\Omega \phi_i \phi_{j,xx} d\Omega = \int_\Gamma \phi_i \phi_{j,x} n_x d\Gamma - \int_\Omega \phi_{i,x} \phi_{j,x} d\Omega.$$

Integral po robu Γ odpade, ker za vsako notranje vozlišče $\mathbf{x}_i$ velja $\phi_i = 0$ na celotnem robu Γ. Z uporabo tega izreka na (2.18) dobimo sistem NDE:

$$\sum_{j=1}^{N} u'_j \int_\Omega \phi_i \phi_j d\Omega + c \sum_{j=1}^{N} u_j \int_\Omega \left[\phi_{i,x}\phi_{j,x} + \phi_{i,y}\phi_{j,y}\right] d\Omega - \int_\Omega \phi_i g d\Omega = 0. \tag{2.19}$$

Integrale označimo v skladu s tradicionalnim poimenovanjem:

$$\begin{aligned} C_{i,j} &= \int_\Omega \phi_i \phi_j d\Omega, \\ K_{i,j} &= c \int_\Omega \left[\phi_{i,x}\phi_{j,x} + \phi_{i,y}\phi_{j,y}\right] d\Omega, \\ f_i &= \int_\Omega \phi_i g d\Omega. \end{aligned} \tag{2.20}$$

Tradicionalno ime za C je *matrika dušenja* (angl. damping matrix). Zdaj lahko sistem (2.19) zapišemo v matrični obliki kot:

$$C\mathbf{u}'(t) + K\mathbf{u}(t) - \mathbf{f}(t) = 0. \tag{2.21}$$

Za diskretizacijo časa imamo na voljo iste metode kot pri MKR. Bistvena razlika pa je, da C ni večkratnik enotske matrike in zato z eksplicitno Eulerjevo metodo dobimo netrivialen sistem:

$$C\mathbf{u}^{(k+1)} = C\mathbf{u}^{(k)} - \Delta t \cdot K\mathbf{u}^{(k)} + \Delta t \cdot \mathbf{f}(t).$$

Z boljšo Crank-Nicolsonovo shemo dobljeni sistem pa ima obliko:

$$C\mathbf{u}^{(k+1)} = C\mathbf{u}^{(k)} - \Delta t \cdot K\frac{\mathbf{u}^{(k)} + \mathbf{u}^{(k+1)}}{2} + \Delta t \cdot \mathbf{f}(t + \frac{\Delta t}{2}),$$

oziroma lepše z neznankami na levi strani:

$$(2C + \Delta t \cdot K)\mathbf{u}^{(k+1)} = (2C - \Delta t \cdot K)\mathbf{u}^{(k)} + 2\Delta t \cdot \mathbf{f}(t + \frac{\Delta t}{2}).$$

Opazimo lahko, da je ta oblika zelo podobna (2.14), zato tudi vsak časovni korak zahteva vse pri MKE omenjene operacije. V splošnem primeru $g = g(x, y, t)$ moramo v vsakem koraku tudi ponovno izračunati vektor $\mathbf{f}$.

Matriki C in K sta tudi tu redki, vendar vsebujeta več neničelnih elementov. Reševanje sistema je nekoliko težje od tistega, dobljenega pri MKR. Simetričnost obeh matrik zagotovimo na podoben način: robnih vozlišč ne oštevilčimo in s tem ukinemo pripadajoče vrstice in stolpce matrik C in K. Vsi elementi stolpcev, ki jih s tem ukinemo, se morajo preseliti v $\mathbf{f}$. Omenimo še, da lahko v primeru trikotnih elementov in linearnih baznih funkcij vse potrebne integrale izračunamo analitično [Buc95].

2.6 Reševanje sistema linearnih enačb

Dobljene sisteme navadnih enačb (linearnih v primeru linearne PDE in linearne diskretizacije) je potrebno tudi rešiti. Dasiravno gre za zelo razvito področje, ki ni predmet tega dela, pa se je za pošteno primerjavo med metodami potrebno zavedati, kako lastnosti linearnega sistema vplivajo na izbor metode za njegovo reševanje in s tem na časovno zahtevnost. Kratek opis nekaterih metod zato ne bo odveč.

2.6.1 Lastnosti sistema

Lastnosti sistema $A\mathbf{x} = \mathbf{b}$ z N enačbami in N neznankami so odvisne od kvadratne matrike A. Rešitev sistema je enolično določena za vsak $\mathbf{b}$, če in samo če je A *nesingularna* in torej ima *inverz*: $AA^{-1} = A^{-1}A = I$. Če je A redka, lahko marsikatera metoda izpusti obravnavanje neničelnih elementov, vendar nekatere izmed metod zahtevajo posebno strukturo le-teh. Nekatere metode zahtevajo tudi posebne matematične lastnosti matrike, kot so:

- *simetričnost*: $A = A^{\mathrm{T}}$ oziroma $A_{i,j} = A_{j,i}$,
- *pozitivna definitnost*: $\forall \mathbf{x} : \mathbf{x}^T A\mathbf{x} > 0$,
- *diagonalna dominantnost po vrsticah*: $\sum_{j=1,j\neq i}^{N} |a_{i,j}| < |a_{i,i}|\,, \quad i \in \{1, \ldots, N\}$.

Težavnost reševanja je odvisna od velikosti sistema, pri dani velikosti pa lahko težavnost napovemo s pomočjo *števila občutljivosti* (angl. condition number) matrike, ki je razmerje med največjo in najmanjšo absolutno lastno vrednostjo. Večje kot je število občutljivosti, bolj občutljiv (slabše pogojen) je sistem in težje je njegovo reševanje. Pri sistemih, nastalih z diskretizacijo PDE, se z večanjem števila vozlišč poleg velikosti sistema navadno veča tudi občutljivost [Hea02, GO93, Ore97].

2.6.2 Neiterativne metode

Metode za reševanje sistema linearnih enačb lahko v grobem razdelimo na tri razrede: *direktne*, *spektralne* in *iterativne*. Podrobnejši opis in teoretično utemeljitev tu predstavljenih metod najdemo v [Hea02, GO93].

Direktne metode

To sta na primer *LU razcep* (angl. LU decomposition) in razcep *Choleskega*. *LU* razcepi poljubno nesingularno matriko A na spodnjetrikotno matriko L, zgornjetrikotno matriko U in permutacijsko matriko P, za katere velja $LU = PA$. Razcep Choleskega razcepi simetrično pozitivno definitno matriko A na spodnje-trikotni in zgornje-trikotni faktor: $A = LL^T$. Časovna zahtevnost razcepa polne matrike, ki bi ga morali v našem primeru

reševanja PDE opraviti le v prvem časovnem koraku, je obakrat $O(N^3)$, v vsakem nadaljnjem koraku pa bi sistem rešili v času $O(N^2)$ z enim vstavljanjem in enim obratnim vstavljanjem.

Težava direktnih metod je, da ne morejo povsem izkoristiti redkosti matrike, ker faktorja vsebujeta veliko več neničelnih elementov kot A – matrika se *zapolni* (angl. fills in). Obstajajo sicer postopki za preurejanje matrike, ki zapolnjevanje omilijo, a se še vedno ne približajo hitrosti iterativnih metod, še posebno za večdimenzijske probleme. Direktne metode je tudi težko paralelizirati.

Spektralne metode

Za nekatere PDE lahko linearne enačbe, ki nastanejo ob diskretizaciji, rešimo s posebnimi metodami, osnovanimi na globalnih periodičnih baznih funkcijah in hitri Fourierovi transformaciji. Če je število vozlišč v vsaki dimenziji potenca števila 2, imajo tovrstne metode zahtevnost $O(N \log N)$. Pri neugodnem številu vozlišč se obnesejo slabše, za splošne oblike domene pa niso uporabne.

2.6.3 Statične iterativne metode

Iterativne metode začnejo reševanje z *začetnim približkom* $\mathbf{x}^{(0)}$, ki ga v vsaki iteraciji popravijo – izboljšajo natančnost. Iteracije ponavljamo, dokler izbrana mera napake ne doseže dovolj majhne vrednosti. Iterativna metoda je *statična*, če lahko opišemo odvisnost novega približka od starega z enostavnim matričnim izrazom, enakim za vse iteracije. Obe tu opisani statični iterativni metodi imata na redkih matrikah z $nnz(A)$ neničelnimi elementi časovno zahtevnost posamezne iteracije $O(nnz(A))$ in prav takšno prostorsko zahtevnost. Paralelizacija obeh metod je enostavna in učinkovita.

Gauss-Seidlova metoda

Gauss-Seidlova metoda v vsaki iteraciji izračuna novo vrednost j-te spremenljivke iz j-te enačbe, pri čemer uporabi najnovejše vrednosti ostalih neznank, kot bi bile te že dokončne. Izvedba programa je enostavna, saj se preprosto sprehodimo čez vektor neznank in sproti prepisujemo stare vrednosti z novimi. Metoda konvergira, če je sistem simetričen in pozitivno definiten in še v nekaterih drugih primerih, vendar je konvergenca počasna.

Metoda SOR

Konvergenco lahko pospešimo tako, da v vsaki iteraciji izračunamo popravek približka $\Delta\mathbf{x}_{\mathbf{GS}}$, ki bi ga predpisala Gauss-Seidlova metoda, ga pomnožimo s konstanto in prište-

jemo staremu približku:

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \omega\Delta\mathbf{x}_{\mathbf{GS}}.$$

Odvisno od parametra ω dobimo *podrelaksacijo* ($\omega < 1$) ali *prerelaksacijo* ($\omega > 1$), po kateri je metoda dobila ime (angl. successive over-relaxation). Veljati mora $0 < \omega < 2$, sicer metoda ne konvergira. Pri optimalnem izboru je konvergenca bistveno hitrejša od Gauss-Seidlove metode, vendar je optimalno vrednost parametra ω praktično nemogoče dobiti drugače kot s poskušanjem.

2.6.4 Metoda konjugiranih gradientov in njene posplošitve

Metoda *konjugiranih gradientov* (angl. conjugate gradients) je optimizacijska iterativna metoda za reševanje sistemov linearnih enačb [Hea02]. Izkorišča posebne lastnosti simetričnih pozitivno definitnih matrik, ki omogočajo v vsaki iteraciji i izbrati od vseh prejšnjih linearno neodvisno smer iskanja $\mathbf{s}_i$, ne da bi morali hraniti več kot eno prejšnjo smer $\mathbf{s}_{i-1}$. V tej iteraciji nato izvedemo *linearno iskanje* v izbrani smeri:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha\mathbf{s}_i.$$

Parameter α je možno izračunati analitično, zato je linearno iskanje končano v enem koraku.

Predpogojevanje

Denimo, da uspemo najti matriko M, ki je čimbolj podobna A in je takšne oblike, da so linearni sistemi s sistemsko matriko M lahko rešljivi. Potem lahko namesto originalnega sistema rešujemo sistem $M^{-1}A\mathbf{x} = \mathbf{y}$; $\mathbf{y} = M^{-1}\mathbf{b}$. Sistem z matriko $M^{-1}A$ je bistveno bolje pogojen (manj občutljiv) od originalnega – tem bolje, čimbolj podobni sta M in A. Za M lahko izberemo na primer *nepopoln razcep* matrike A – algoritmu za LU razcep omejimo vpisovanje neničelnih vrednosti na mesta, kjer so v A ničle [Hea02].

Metoda konjugiranih gradientov s predpogojevanjem

Metoda konjugiranih gradientov s predpogojevanjem (angl. preconditioning) začne z začetnim približkom $\mathbf{x}_0$. Izračuna $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ in $\mathbf{s}_0 = M^{-1}\mathbf{r}_0$, nakar do konvergence ponavlja naslednje korake:

1. $\alpha_i = \mathbf{r}_i^T M^{-1}\mathbf{r}_i / \mathbf{s}_i^T A\mathbf{s}_i$,
2. $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i\mathbf{s}_i$,
3. $\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i A\mathbf{s}_i$,
4. $\beta_{i+1} = \mathbf{r}_{i+1}^T M^{-1}\mathbf{r}_{i+1} / \mathbf{r}_i^T M^{-1}\mathbf{r}_i$,
5. $\mathbf{s}_{i+1} = M^{-1}\mathbf{r}_{i+1} + \beta_{i+1}\mathbf{s}_i$.

V povprečju vsaka iteracija zmanjša napako za faktor $\frac{\sqrt{cond(M^{-1}A)}-1}{\sqrt{cond(M^{-1}A)}+1}$, kjer je $cond(A)$ število občutljivosti [Hea02].

Časovna zahtevnost vsake iteracije je $O(nnz(A))$, k čemur moramo prišteti še reševanje sistema oblike $M\mathbf{z} = \mathbf{c}$. Tega je potrebno rešiti enkrat pred prvo iteracijo in v koraku 4 vsake iteracije, v koraku 1 pa uporabimo rešitev iz prejšnje iteracije.

Za učinkovito paralelizacijo moramo ustrezno izbrati način predpogojevanja – če uporabimo nepopoln razcep, se bo izračun $M^{-1}\mathbf{r}_{i+1}$ izvajal popolnoma zaporedno [SZ02]. Ostali koraki iteracije niso problematični.

Metoda BiCGSTAB

Za nesimetrične sisteme obstaja vrsta posplošitev metode konjugiranih gradientov, kot so *kvadratni konjugirani gradienti* (CGS, angl. conjugate gradient squared), *bikonjugirani gradienti* in *bikonjugirani stabilizirani gradienti* (BiCGSTAB). Slednja metoda najprej izračuna $\tilde{\mathbf{r}} = \mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, nakar do konvergence ponavlja [BBC+94]:

1. $\rho_{i-1} = \tilde{\mathbf{r}}^{\mathrm{T}}\mathbf{r}_{i-1}$,
2. če $\rho_{i-1} = 0$, metoda odpove,
3. če $i = 1$, potem $\mathbf{p}_i = \mathbf{r}_{i-1}$, sicer:
$$\beta_{i-1} = \frac{\rho_{i-1}/\rho_{i-2}}{\alpha_{i-1}/\omega_{i-1}},$$
$$\mathbf{p}_i = \mathbf{r}_{i-1} + \beta_{i-1}(\mathbf{p}_{i-1} - \omega_{i-1}\mathbf{v}_{i-1}),$$
4. reši $M\hat{\mathbf{p}} = \mathbf{p}_i$,
5. $\mathbf{v}_i = A\hat{\mathbf{p}}$,
6. $\alpha_i = \rho_{i-1}/\tilde{\mathbf{r}}^{\mathrm{T}}\mathbf{v}_i$,
7. $\mathbf{s} = \mathbf{r}_{i-1} - \alpha_i\mathbf{v}_i$,
8. preveri normo $\mathbf{s}$; če je dovolj majhna, $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\hat{\mathbf{p}}$ in končaj,
9. reši $M\hat{\mathbf{s}} = \mathbf{s}$,
10. $\mathbf{t} = A\hat{\mathbf{s}}$,
11. $\omega_i = \mathbf{t}^{\mathrm{T}}\mathbf{s}/\mathbf{t}^{\mathrm{T}}\mathbf{t}$,
12. $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\hat{\mathbf{p}} + \omega_i\hat{\mathbf{s}}$,
13. $\mathbf{r}_i = \mathbf{s} - \omega_i\mathbf{t}$,
14. preveri normo $\mathbf{r}_i$; če je dovolj majhna, končaj,
15. če $\omega_i = 0$, metoda odpove.

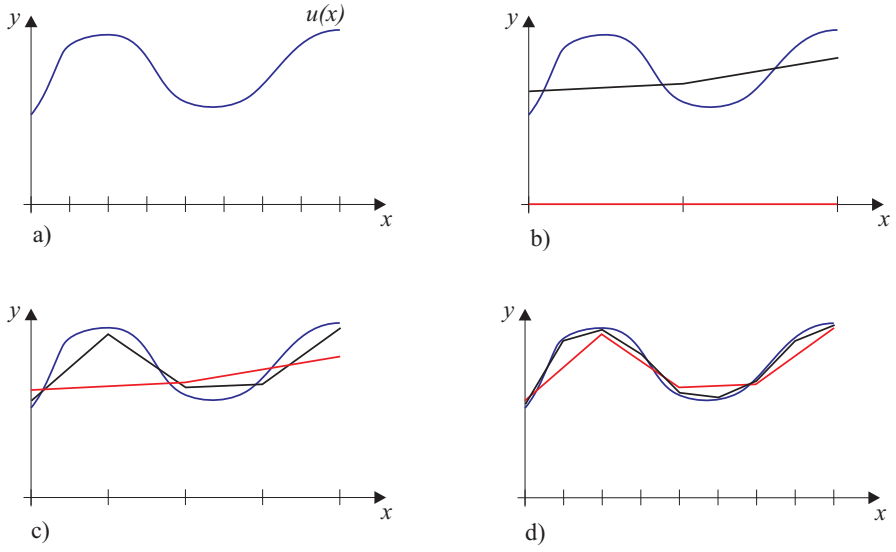
Opazimo lahko, da metoda potencialno odpove. Vprašanji, v katerih primerih se to zgodi in kako hitro metoda konvergira, če ne odpove, sta slabo teoretično raziskani. V praksi za mnoge sisteme, nastale pri diskretizaciji PDE, deluje zanesljivo in z uporabo dobrega

načina predpogojevanja konvergira hitro [Bic, BBC+94]. Časovna zahtevnost posamezne iteracije je nekajkrat večja, vendar asimptotično enaka kot pri metodi konjugiranih gradientov.

2.6.5 Metode multigrid

Pri sistemih linearnih enačb, ki izvirajo iz diferencialnih enačb, je vektor $\mathbf{x}$ ravno diskretizirana funkcija, ki reši diferencialno enačbo, zato je smiselno opazovati rešitev in napako v *frekvenčnem prostoru*. Statične iterativne metode visokofrekvenčne komponente napake zelo hitro zmanjšajo (pravimo, da napako *zgladijo*), zato jih včasih imenujemo kar *zglajevalke* (angl. smoothers). Nizkofrekvenčne komponente jim povzročajo preglavice in so razlog za počasno konvergenco [Hea02].

Nizkofrekvenčne komponente se ne spremenijo dosti, če zmanjšamo natančnost diskretizacije – statične iterativne metode jih brez potrebe računajo v polni ločljivosti. Pač pa komponente, ki so bile pri fini diskretizaciji nizkofrekvenčne, z zmanjšanjem ločljivosti postanejo visokofrekvenčne, zato več ne povzročajo težav s konvergenco statičnih iterativnih metod. Na grobi mreži z zglajevalko hitro odpravimo te komponente, dobljeno rešitev *interpoliramo* nazaj na polno ločljivost in spet z zglajevalko hitro odpravimo še visokofrekvenčne komponente napake. Razne metode, katerih bistvo je takšno računanje na različnih ločljivostih, imenujemo *multigrid* [Wes91], primer, kjer so uporabljene tri stopnje ločljivosti, pa prikazuje slika 2.6.



Slika 2.6: Približevanje rešitvi $f(x)$ (a) na treh stopnjah ločljivosti (b,c,d). Vsakokrat je začetni približek obarvan rdeče, dobljena rešitev pa črno. Na najbolj grobi mreži statična iterativna metoda hitro najde pravi odmik po y, ki bi na polni ločljivosti upočasnil konvergenco.

Z rešitve $\mathbf{x}_f$ na finejši mreži na grobo $\mathbf{x}_c$ prehajamo z linearnim operatorjem *omejevanja* (angl. restriction) I_f^c: $\mathbf{x}_c = I_f^c \mathbf{x}_f$, pa tudi sistemsko matriko na grobi mreži tvorimo na enak način: $A_c = I_f^c A_f$. Obraten prehod naredimo z operatorjem *interpolacije*, ki je običajno transponiran operator omejevanja [Wes91]: $\mathbf{x}_f = I_c^f \mathbf{x}_c = (I_f^c)^{\mathrm{T}} \mathbf{x}_c$.

Za zglajevalko tipično uporabimo nekaj iteracij Gauss-Seidlove metode, omejevanje in interpolacija pa zahtevata hierarhijo različno finih mrež. Z dobrim vzorcem prehajanja med mrežami, na primer algoritmom *polni multigrid* [GO93], lahko dosežemo idealno obnašanje – število potrebnih iteracij ni odvisno od velikosti sistema, zato je časovna zahtevnost metode enaka zahtevnosti ene iteracije, to je $O(N \log N)$. Paralelizacija je skoraj tako učinkovita kot paralelizacija zglajevalke, saj ta predstavlja največji delež računanja v metodah multigrid [ŠT03, DHL03].

Metode multigrid se izredno dobro obnesejo na sistemih, nastalih z MKR [ŠT04]. Tudi pri MKE so uspešne, le pravilna izvedba omejevanja in interpolacije zahteva nekaj več pazljivosti. Mreže elementov za MKE se navadno generira iz začetne grobe mreže z večkratnim deljenjem elementov na manjše, zato dobimo hierarhijo različno gostih mrež.

Žal je uporaba metod multigrid težavna na sistemih, nastalih z mreže prostimi metodami, saj nimamo hierarhije mrež in pri nekaterih različicah tudi ne naravne definicije omejevanja in interpolacije [LOS04]. Nadaljnja težava je iskanje primerne zglajevalke, saj pogosto dobimo nesimetrične in nedefinitne sisteme, na katerih Gauss-Seidlova metoda ne deluje. Metoda BiCGSTAB ni primerna za zglajevalko, ker ne odpravlja prvenstveno visokofrekvenčnih komponent napake.

Poglavje 3

Mreže proste metode

V tem poglavju predstavimo različne vrste mreže prostih metod in podrobno opišemo matematično ozadje izbranih različic. Najprej obravnavamo bazne funkcije, po katerih se MPM najbolj ločijo od MKE. Konstruiramo interpolacijske bazne funkcije in aproksimacijske bazne funkcije, dobljene z metodo gibljivih najmanjših kvadratov. Pri slednjih se posebej posvetimo določanju nosilne domene in zagotavljanju zveznosti. Obravnavane bazne funkcije ilustriramo tudi s primeri v eni in dveh dimenzijah.

V nadaljevanju poglavja pojasnimo, kako drugačnost baznih funkcij v primerjavi s tistimi pri MKE vpliva na pretvorbo PDE v sistem navadnih enačb. Glede na način konstrukcije baznih funkcij in metodo vsiljevanja robnih pogojev razdelimo MPM na več razredov. Vse pridobljeno znanje uporabimo za izpeljavo sistema navadnih enačb za reševanje dvodimenzijske difuzijske enačbe z mreže prosto lokalno Petrov-Galerkinovo metodo.

3.1 Bazne funkcije

Podobno kot MKE tudi MPM poskusno funkcijo sestavi kot linearno kombinacijo baznih funkcij:

$$\hat{u}(\mathbf{x}) = \sum_{j=1}^{N} u_j \phi_j(\mathbf{x}) = \mathbf{u}^{\mathrm{T}} \mathbf{\Phi}(\mathbf{x}), \tag{3.1}$$

ki odsekoma bodisi *aproksimira* ali *interpolira* vrednosti vozliščnih parametrov $\mathbf{u}$ v vozliščih $\{\mathbf{x}_I, I = 1 \dots N\}$ [FM04, Liu03, AS02]. Za primerjavo omenimo, da gre pri MKE vedno za interpolacijo, pri MPM pa je $\hat{u}$ dobljena z aproksimacijo ali interpolacijo. Podobno kot pri MKE poskusna funkcija $\hat{u}$ ni odvisna le od vrednosti vozliščnih parametrov $\mathbf{u}$, marveč tudi od položajev vozlišč – ta odvisnost se mora očitno skrivati v formulaciji baznih funkcij. Posamična bazna funkcija ϕ_i je enaka aproksimaciji ali interpolaciji vrednosti 1 v vozlišču $\mathbf{x}_i$ in 0 v vseh ostalih vozliščih, to je poskusni funkciji z vozliščnimi parametri z edino enico na i-tem mestu: $\mathbf{u} = [0, \dots, 0, 1, 0, \dots, 0]$.

Bazne funkcije lahko formuliramo na različne načine, pri čemer si želimo izpolniti naslednje cilje [Liu03]:

1. Omogočiti razmeroma *prosto izbiro vozlišč*, oziroma vsaj manj omejeno kot pri MKE. Pri tem naj bo algoritem za konstrukcijo baznih funkcij stabilen – ne sme odpovedati pri morebitni manj ugodni razporeditvi vozlišč.
2. Bazne funkcije naj bodo zadostnega *reda konsistentnosti* – zmožne naj bodo točno predstaviti polinome tega reda. To zagotavlja, da z MPM dobljena rešitev konvergira k pravi, ko zgoščamo vozlišča.
3. Algoritem za konstrukcijo baznih funkcij naj bo računsko učinkovit.
4. Bazne funkcije naj imajo lokalni nosilec. Tako zmanjšamo število neničelnih elementov v matriki končnega linearnega sistema in s tem zahtevnost njegovega reševanja.
5. Po možnosti naj imajo bazne funkcije Kroneckerjevo δ lastnost, kar omogoči enostavno vsiljevanje robnih pogojev (za definicijo te lastnosti glej enačbo 2.15 na strani 27).
6. Ugodno je, če je poskusna funkcija ne glede na postavitev vozlišč zvezna po celotni domeni Ω, saj je povečini zvezna tudi točna rešitev PDE. Nekatere različice MPM z nezveznimi baznimi funkcijami sploh ne dajo uporabnih rezultatov.

Kot bomo videli, ni znana nobena vrsta baznih funkcij, ki bi bila zvezna in hkrati imela Kroneckerjevo δ lastnost. Za interpolacijske bazne funkcije se največkrat uporabi bodisi *polinome*, *radialne funkcije* ali kombinacijo obeh [BKO+96]. Med aproksimacijskimi metodami je najbolj razširjena aproksimacija z *gibljivimi najmanjšimi kvadrati* (angl. moving least squares, MLS) [LS81], ki jo med drugim uporabljajo elementov prosta metoda Galerkina (EPG, angl. element free Galerkin method, EFG) [BLG94] in mreže proste lokalne Petrov-Galerkinove metode (MLPG, angl. meshless local Petrov-Galerkin methods) [AS02]. Za simulacije toka tekočine in nekatere druge aplikacije se uporabljajo še drugi tipi baznih funkcij, na primer *smoothed particle hydrodynamics* (slov. hidrodinamika z zglajenimi delci) [Luc77] in sorodne *reproducing kernel particle method* (slov. metoda delcev, ki ohranjajo jedro) [LAJ93].

V nadaljevanju najprej definiramo temeljna pojma: nosilno domeno točke in nosilna vozlišča, nato pa za interpolacijo in za aproksimacijo z gibljivimi najmanjšimi kvadrati pokažemo konstrukcijo baznih funkcij in primere le-teh. Prikazane bazne funkcije primerjamo med seboj s pomočjo analize nekaterih matematičnih lastnosti.

3.1.1 Nosilna domena točke in nosilna vozlišča

Lokalne nosilce baznih funkcij zagotovimo z lokalno aproksimacijo oziroma interpolacijo vozliščnih parametrov: vrednost poskusne funkcije $\hat{u}(\mathbf{x})$ naj bo aproksimacija oziroma interpolacija le tistih vozliščnih parametrov, katerih vozlišča ležijo v bližini točke $\mathbf{x}$. Ostala vozlišča in njihovi parametri naj na vrednost $\hat{u}(\mathbf{x})$ ne vplivajo, torej moramo njihove

bazne funkcije definirati tako, da bodo imele v točki $\mathbf{x}$ vrednost 0 [BKO+96].

Za vsako točko $\mathbf{x}$ definirajmo neko okolico, ki jo poimenujemo *nosilna domena* (angl. support domain) točke in označimo z $\Omega_S(\mathbf{x})$. Množico vozlišč, ki ležijo znotraj nosilne domene točke, imenujmo *nosilna vozlišča* točke. Vrednost poskusne funkcije je aproksimacija oziroma interpolacija vozliščnih parametrov nosilnih vozlišč.

Nosilna domena točke $\mathbf{x}$ je torej definirana kot področje (v dveh dimenzijah navadno krog, elipsa ali pravokotnik), katerega vozlišča vplivajo na vrednost $\hat{u}(\mathbf{x})$. Nosilna domena v MPM mora biti ravno prav velika, da vanjo pade za interpolacijo oziroma aproksimacijo primerno število vozlišč. Ker so vozlišča v celotni domeni Ω običajno različno gosta, moramo velikost nosilne domene določevati lokalno [Liu03].

3.1.2 Interpolacija

Parametre vozlišč interpoliramo tako, da poskusno funkcijo definiramo kot:

$$\hat{u}(\mathbf{x}) = \sum_{i=1}^{n} p_i(\mathbf{x}) a_i(\mathbf{x}) = \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{a}(\mathbf{x}), \tag{3.2}$$

kjer je $\mathbf{p}$ vektor funkcij, uporabljenih za interpolacijo, $\mathbf{a}$ vektor njihovih koeficientov, n pa število funkcij v $\mathbf{p}$. Za nosilna vozlišča izberemo kar n najbližjih vozlišč, ki jih bomo označili z $\mathbf{x}_S = \{\mathbf{x}_{S_i}, i = 1 \dots n\}$. Za funkcije se najpogosteje uporablja bodisi *monome* v ustreznem številu dimenzij:

$$\begin{aligned}
\mathbf{p}^{\mathrm{T}}(x) &= [1, x, x^2, \dots], \\
\mathbf{p}^{\mathrm{T}}(x, y) &= [1, x, y, x^2, y^2, xy, \dots], \\
\mathbf{p}^{\mathrm{T}}(x, y, z) &= [1, x, y, z, x^2, y^2, z^2, xy, xz, yz, \dots],
\end{aligned}$$

radialne funkcije ali kombinacijo obojih. Radialne funkcije definiramo tako, da ima j-ta radialna funkcija vrh v vozlišču $\mathbf{x}_j$, v drugih točkah pa je njena vrednost odvisna le od razdalje $r_j = ||\mathbf{x}_j - \mathbf{x}||_2$ do vozlišča. Primera radialnih funkcij sta *multikvadrika* (angl. multiquadrics, MQ) in *Gaussova radialna funkcija* (EXP) [Liu03, ICT04]:

$$p_{j\,MQ}(r_j) = \left(r_j^2 + C^2\right)^q, \qquad p_{j\,EXP}(r_j) = e^{-cr_j^2}.$$

Bazne funkcije

Interpolacijske bazne funkcije izrazimo tako, da (3.2) pretvorimo v (3.1). Vektor koeficientov $\mathbf{a}$ je enak za vse točke z istimi nosilnimi vozlišči – v teh točkah za $\hat{u}$ uporabimo isti *odsek* interpolacije. Vrednosti $\mathbf{a}$ dobimo tako, da v vseh nosilnih vozliščih $\mathbf{x}_S$ desno stran (3.2) izenačimo s parametrom vozlišča [Liu03]. Dobimo sistem:

$$P(\mathbf{x}_S)\mathbf{a}(\mathbf{x}_S) = \mathbf{u}_S,$$

kjer *momentna matrika* P vsebuje vrednosti $\mathbf{p}$ v $\mathbf{x}_S$:

$$P(\mathbf{x}_S) = \begin{bmatrix} \mathbf{p}^{\mathrm{T}}(\mathbf{x}_{S_1}) \\ \mathbf{p}^{\mathrm{T}}(\mathbf{x}_{S_2}) \\ \vdots \\ \mathbf{p}^{\mathrm{T}}(\mathbf{x}_{S_n}) \end{bmatrix}$$

Poudarimo, da so P, $\mathbf{a}$ in $\mathbf{u}_S$ odvisni od točke $\mathbf{x}$ zato, ker slednja določa množico nosilnih vozlišč $\mathbf{x}_S$.

Predpostavimo, da je interpolacija enolično določena. V tem primeru je P nesingularna in lahko njen inverz uporabimo v (3.2):

$$\hat{u}(\mathbf{x}) = \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{a}(\mathbf{x}_S) = \mathbf{p}^{\mathrm{T}}(\mathbf{x})P^{-1}(\mathbf{x}_S)\mathbf{u}_S = \sum_{j=1}^{n} \phi_j(\mathbf{x})u_{S_j}.$$

Velikost in oblika nosilne domene ne vplivata na poskusno funkcijo, pomembna je le množica nosilnih vozlišč [Liu03].

Pri reševanju PDE z MPM moramo znati izračunati bazne funkcije in njihove odvode v poljubni točki. Bazne funkcije lahko zapišemo kot:

$$\mathbf{\Phi}(\mathbf{x}) = \mathbf{p}^{\mathrm{T}}(\mathbf{x})P^{-1}(\mathbf{x}_S),$$

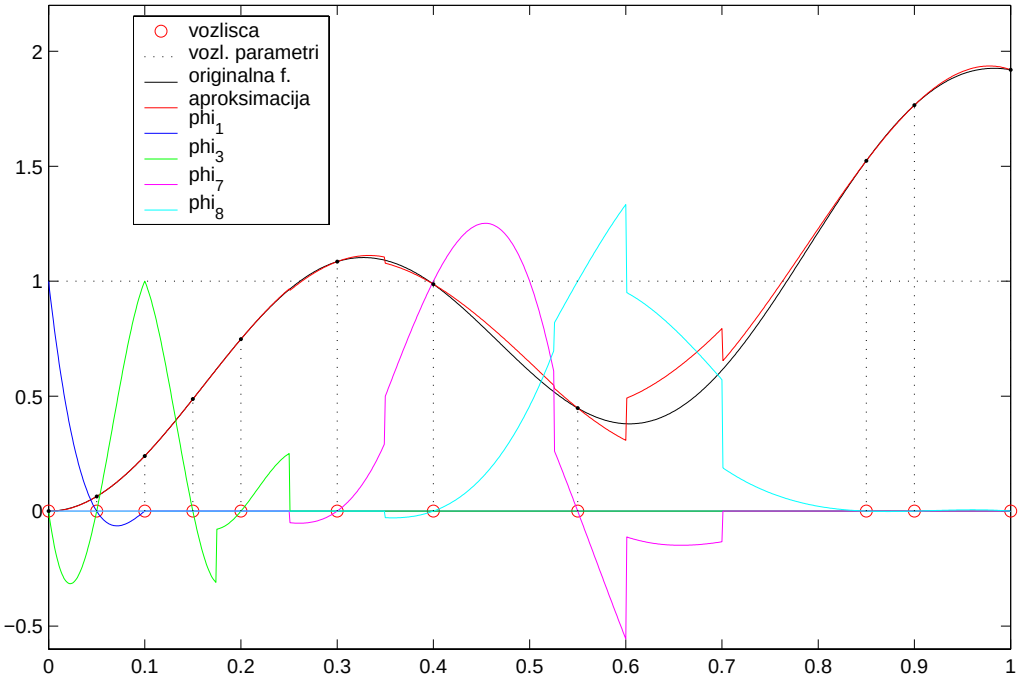
odvajanje pa je trivialno, ker P ni odvisna od $\mathbf{x}$, razen tam, kjer se spremeni $\mathbf{x}_S$.

Slika 3.1 prikazuje polinomsko interpolacijo funkcije $f(x) = \sin^2(5x) + x^2$ in nekaj pripadajočih baznih funkcij. Za to interpolacijo so bili uporabljeni monomi stopenj od 0 do 3, torej $\mathbf{p}^{\mathrm{T}} = [1, x, x^2, x^3]$. Vozlišča so označena z rdečimi krogci. Nosilna vozlišča vsake točke so bila najbližja štiri vozlišča.

Lastnosti interpolacijskih baznih funkcij

Kot smo že povedali, i-ta bazna funkcija MPM vedno interpolira oziroma aproksimira vrednost 1 v i-tem vozlišču in 0 v vseh ostalih. Od tu neposredno sledi, da imajo interpolacijske bazne funkcije Kroneckerjevo δ lastnost, kar omogoči enostavno vsiljevanje Dirichletovih robnih pogojev [FM04]. Parametru robnega vozlišča zgolj predpišemo vrednost robnega pogoja in dobljena rešitev $\hat{u}$ bo imela v robnih vozliščih pravo vrednost. Na sliki 3.1 je robni pogoj $f(0) = 0$, zato je pripadajoča bazna funkcija ϕ_1 pomnožena z 0.

Na sliki 3.1 takoj opazimo nezveznost tako baznih funkcij kot poskusne funkcije, kar je posledica interpolacije po odsekih. Na sliki je največja nezveznost pri $x = 0.6$. Za točke levo od 0.6 so nosilna vozlišča $0.3, 0.4, 0.55$ in 0.85 in je vrednost $\hat{u}$ enaka polinomu, ki seka funkcijo v teh vozliščih. Ob prehodu čez točko nezveznosti 0.6 vozlišče 0.3 izstopi iz



Slika 3.1: Polinomska interpolacija funkcije $f(x) = \sin^2(5x) + x^2$ in nekatere bazne funkcije.

množice nosilnih vozlišč, vanjo pa vstopi vozlišče 0.9, zato je od tu naprej uporabljen drug interpolacijski polinom. V enodimenzionalnem primeru bi se nezveznosti lahko izognili tako, da bi se nosilna vozlišča, uporabljena za interpolacijo, menjala v vozliščih namesto med njimi. V našem primeru bi vozlišče 0.3 zamenjali z 0.9 že ob prehodu čez vozlišče 0.55. Tam imata polinoma, uporabljena za levi in desni odsek, isto vrednost, saj obe interpolirata vrednost v tem vozlišču. V dvo- in tridimenzionalnem primeru pa zveznosti ne moremo enostavno zagotoviti [Liu03].

Interpolacijske bazne funkcije zmorejo točno predstaviti vsako funkcijo, ki jo lahko izrazimo kot linearno kombinacijo funkcij v $\mathbf{p}$, saj bo imela ta že sama po sebi obliko enačbe 3.2. Prva pomembna posledica tega dejstva je, da imajo bazne funkcije tak polinomski red konsistentnosti, kot je najvišji red monomov, vsebovanih v $\mathbf{p}$. Za razliko od polinomskih radialne bazne funkcije nimajo niti konsistentnosti reda 0, zato ne morejo natančno predstaviti niti konstantne funkcije in se pogosto kombinirajo z monomi nizkega reda. Druga posledica je, da lahko v $\mathbf{p}$ vključimo kako posebno funkcijo, kadar iz vnaprejšnjega znanja o rešitvi sklepamo, da bi bilo to koristno [AS02].

Singularnost momentne matrike

Momentna matrika P je singularna natanko takrat, kadar interpolacija ni enolično določena [Liu03]. Za kvadratno polinomsko interpolacijo v dveh dimenzijah potrebujemo šest

nosilnih vozlišč, saj je vektor monomov $\mathbf{p}^{\mathrm{T}} = [1, x, y, x^2, y^2, xy]$, poleg števila vozlišč pa je pomemben tudi njihov medsebojni položaj. Lahko se zgodi, da so vozlišča razporejena v dve vodoravni premici in imajo vsa skupaj le dve različni y-koordinati, zato lahko skoznje potegnemo neskončno različnih parabol v y-smeri. To pomeni, da interpolacijska funkcija ni enolično določena in je matrika P singularna.

Iz singularnosti se lahko rešimo na več načinov [Liu03], izmed katerih pa nobeden ni idealen:

- Naključno premaknemo nekatera vozlišča in upamo, da bo P postala nesingularna. S tem lahko povzročimo singularnost P v kaki drugi točki. Če ne poznamo vnaprej seznama točk, v katerih bo potrebno izračunati P, ne moremo biti nikoli prepričani, da smo problem zares rešili.
- Uvedemo lokalni koordinatni sistem, ki je rotiran glede na globalnega. Tako lahko odpravimo singularnost v mnogih, ne pa v vseh primerih.
- S triangularizacijo matrike P ugotovimo, kateri monomi so problematični, kot je bil v zgoraj omenjenem primeru y^2. Le-te odstranimo iz $\mathbf{p}$ in hkrati iz množice nosilnih vozlišč odstranimo ustrezno število vozlišč [LG03]. Postopek zaenkrat ni dovolj razvit, da bi reševal vse primere.

Singularnosti se lahko povsem ognemo z uporabo radialnih funkcij ali kombinacije teh in monomov stopenj 0 in 1. Za te primere je dokazano, da P ne more biti singularna, če parametre radialnih funkcij izberemo znotraj znanih varnih intervalov [Wen98].

3.1.3 Aproksimacija z gibljivimi najmanjšimi kvadrati

Aproksimacija z *gibljivimi najmanjšimi kvadrati* (angl. moving least squares, MLS), imenovana tudi *lokalno utežena regresija* (angl. locally weighted regression) [LS81, Cle93], izvira iz statistike in se je kot zelo uporabna izkazala v MPM. Osnovna ideja je v vsaki točki aproksimirati dane podatke lokalno, z uporabo razmeroma majhnega števila najbližjih vozlišč, to je tistih v nosilni domeni, in le nekaj (m) monomov:

$$\hat{u}(\mathbf{x}) = \sum_{i=1}^{m} p_i(\mathbf{x}) a_i(\mathbf{x}) = \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{a}(\mathbf{x}), \tag{3.3}$$

kjer je $\mathbf{p}$ vektor vseh monomov stopenj nič in ena (na primer v dveh dimenzijah $\mathbf{p}^{\mathrm{T}}(\mathbf{x}) = [1, x, y]$) ali od nič do dva (na primer $\mathbf{p}^{\mathrm{T}}(\mathbf{x}) = [1, x, y, x^2, y^2, xy]$). Tu je $\mathbf{a}$ vektor koeficientov, ki so funkcije $\mathbf{x}$, ker so koeficienti za vsak x drugačni [Liu03].

Eden od ciljev MLS je zagotoviti zveznost aproksimacije, zato morajo vozlišča vstopati in izstopati iz nosilne domene zvezno – ko se vozlišče približuje robu nosilne domene, ga moramo vse manj upoštevati. To zagotovimo z *utežno funkcijo* v obliki klobuka, ki ima vrh v $\mathbf{x}$ in do roba nosilne domene pade na 0. Pomemben je torej položaj nosilnih vozlišč znotraj nosilne domene, s tem pa tudi oblika in velikost nosilne domene.

V [Liu03] je priporočena uporaba funkcije:

$$W_I(\mathbf{x}) = W\left(\frac{||\mathbf{x}_I - \mathbf{x}||_2}{d_S(\mathbf{x})/2}\right), \quad \text{kjer je } W(d) = \begin{cases} 1 - 6d^2 + 8d^3 - 3d^4 & d \leq 1 \\ 0 & d > 1, \end{cases} \tag{3.4}$$

$d_S(\mathbf{x})$ premer nosilne domene točke $\mathbf{x}$ in $\mathbf{x}_I$ vozlišče, katerega utež izraža W_I. Utežna funkcija ter njen prvi in drugi odvod so zvezni in na robovih nosilne domene enaki 0. Z definicijo utežne funkcije implicitno izberemo tudi obliko nosilne domene, na primer krog, pravokotnik itd. V primeru funkcije (3.4) je nosilna domena krog.

Izračun koeficientov

Koeficiente $\mathbf{a}$ lahko izračunamo tako, da definiramo funkcional utežene razlike med vozliščnimi parametri in aproksimacijo (3.3) ter ga odvajamo po $\mathbf{a}$ [FM04, Liu03]. Tu bomo predstavili ekvivalentno izpeljavo, podobno tisti za *linearne najmanjše kvadrate* (angl. linear least squares) v [Hea02].

Brez uteži se problema lotimo tako, da nastavimo podoben sistem enačb kot za interpolacijo, vendar z nekvadratno momentno matriko P:

$$P_{(n\times m)}\mathbf{a}_{(m\times 1)} = \begin{bmatrix} \mathbf{p}^{\mathrm{T}}(\mathbf{x}_1) \\ \mathbf{p}^{\mathrm{T}}(\mathbf{x}_2) \\ \vdots \\ \mathbf{p}^{\mathrm{T}}(\mathbf{x}_n) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_m \end{bmatrix} \approx \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \mathbf{u}_{S(n\times 1)}, \tag{3.5}$$

kjer je n število nosilnih vozlišč, u_S so njihovi parametri, podpisani oklepaji pa označujejo velikosti matrik. Veljati mora $n > m$ in P mora imeti *poln rang*, to je m, da bo sistem *predeterminiran* in ga bomo rešili približno v smislu najmanjših kvadratov. Pri $n = m$ dobimo interpolacijo, pri $n < m$ ali $\mathrm{rang}(P) < m$ pa *poddeterminiran* sistem.

Sistem lahko rešimo z *metodo normalnih enačb* [Hea02]: radi bi minimizirali kvadrat evklidske norme ostanka $\mathbf{r} = \mathbf{u}_S - P\mathbf{a}$, to je:

$$||\mathbf{r}||_2^2 = \mathbf{r}^{\mathrm{T}}\mathbf{r} = (\mathbf{u}_S - P\mathbf{a})^{\mathrm{T}}(\mathbf{u}_S - P\mathbf{a}) = \mathbf{u}_S^{\mathrm{T}}\mathbf{u}_S - 2\mathbf{a}^{\mathrm{T}}P^{\mathrm{T}}\mathbf{u}_S + \mathbf{a}^{\mathrm{T}}P^{\mathrm{T}}P\mathbf{a}.$$

Izraz odvajamo po $\mathbf{a}$ in odvod izenačimo z 0:

$$2P^{\mathrm{T}}P\mathbf{a} - 2P^{\mathrm{T}}\mathbf{u}_S = 0, \qquad \text{oziroma} \qquad P^{\mathrm{T}}P\mathbf{a} = P^{\mathrm{T}}\mathbf{u}_S,$$

torej moramo rešiti simetričen kvadraten sistem velikosti $m \times m$, kar zahteva približno $m^2n/2 + m^3/6$ množenj in približno toliko seštevanj. Alternativno lahko sistem rešimo s katero od boljših metod, na primer *Householderjevo* [Hea02], ki ob nekaj večjem številu operacij ($m^2n - m^3/3$ množenj in približno toliko seštevanj) da robustnejšo rešitev, manj odvisno od občutljivosti matrike P. Žal se nam, kot bomo pokazali na strani 47, s Householderjevo metodo zatakne pri formulaciji odvodov baznih funkcij.

Uteži vpeljemo preprosto tako, da vsako enačbo v (3.5) vključno s pripadajočo desno stranjo pomnožimo s $\sqrt{W}$. Pri približnem reševanju sistema bo enačba z manjšo utežjo manj upoštevana, ker manj vpliva na ostanek. V kvadratu ostanka bo tako vsak člen utežen ravno z W. Sistem (3.5) z dodajanjem uteži spremenimo v:

$$P_W \mathbf{a} = \begin{bmatrix} \sqrt{W_1(\mathbf{x})}\mathbf{p}^{\mathrm{T}}(\mathbf{x}_1) \\ \sqrt{W_2(\mathbf{x})}\mathbf{p}^{\mathrm{T}}(\mathbf{x}_2) \\ \vdots \\ \sqrt{W_n(\mathbf{x})}\mathbf{p}^{\mathrm{T}}(\mathbf{x}_n) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_m \end{bmatrix} \approx V\mathbf{u}_S, \tag{3.6}$$

kjer je $V_{(n\times n)}$ diagonalna matrika korenov uteži:

$$V_{i,j} = \begin{cases} \sqrt{W_i(\mathbf{x})} & i = j \\ 0 & \text{sicer.} \end{cases}$$

Z metodo normalnih enačb dobimo končni sistem za koeficiente aproksimacije MLS:

$$P_W^{\mathrm{T}} P_W \mathbf{a} = P_W^{\mathrm{T}} V \mathbf{u}_S \qquad \text{oziroma} \qquad A_{(m\times m)}\mathbf{a} = B_{(m\times n)}\mathbf{u}_S, \tag{3.7}$$

kjer sta novi matriki A in B:

$$A_{i,j} = \sum_{k=1}^{n} W_k(\mathbf{x}) p_i(\mathbf{x}_k) p_j(\mathbf{x}_k), \qquad B_{i,j} = W_j(\mathbf{x}) p_i(\mathbf{x}_j).$$

Bazne funkcije

Bazne funkcije izrazimo tako, da rešitev sistema (3.7) vstavimo v (3.3):

$$\hat{u}(\mathbf{x}) = \mathbf{p}^{\mathrm{T}}(\mathbf{x}) A^{-1}(\mathbf{x}) B(\mathbf{x}) \mathbf{u}_S = \boldsymbol{\Phi}(\mathbf{x}) \mathbf{u}_S,$$

torej

$$\phi_{j(1\times 1)}(\mathbf{x}) = \sum_{i=1}^{m} p_i(\mathbf{x}) \left[A^{-1}(\mathbf{x}) B(\mathbf{x})\right]_{i,j} = \mathbf{p}^{\mathrm{T}}_{(1\times m)} A^{-1}_{(m\times m)} B_{1\ldots m, j(m\times 1)}. \tag{3.8}$$

Pri ovrednotenju izraza se seveda izognemo računanju matričnega inverza in vsak produkt matrik, v katerem je levi faktor inverz, zamenjamo z reševanjem ustreznega sistema.

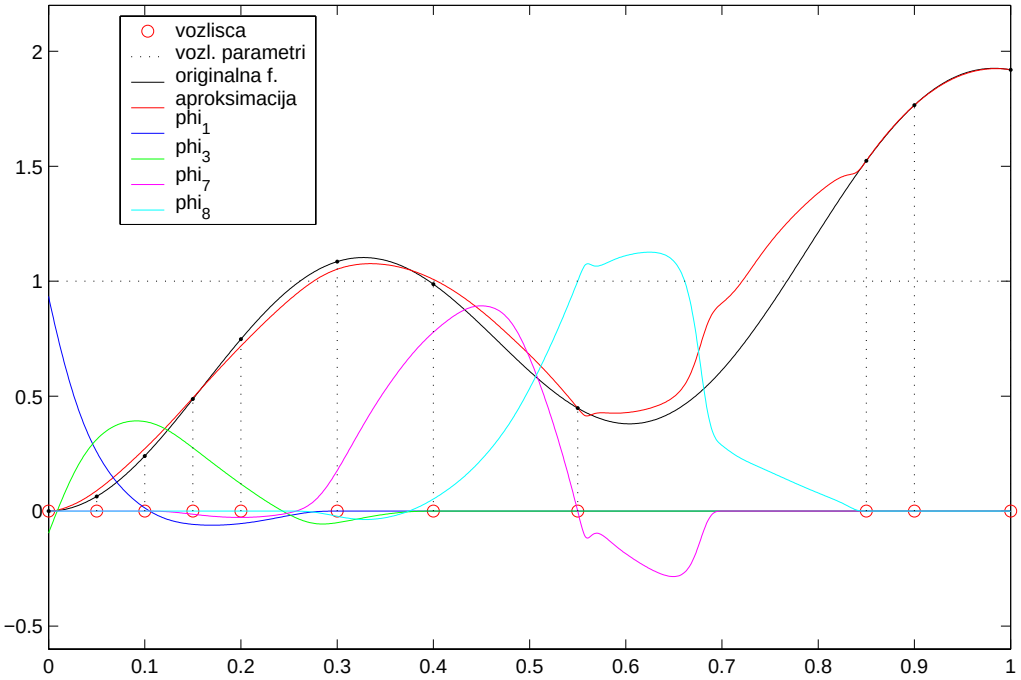
Slika 3.2 prikazuje aproksimacijo funkcije $f(x) = \sin^2(5x) + x^2$ z istimi vozlišči kot na sliki 3.1. Uporabljeni so monomi $\mathbf{p}^{\mathrm{T}} = [1, x, x^2]$, dolžina nosilne domene je 0.6.

Odvode lahko izrazimo tako, da bazne funkcije zapišemo s pomožnim vektorjem $\mathbf{g}_{(m\times 1)}$ [Liu03]:

$$\boldsymbol{\Phi}(\mathbf{x}) = \mathbf{g}^{\mathrm{T}}(\mathbf{x}) B(\mathbf{x}), \qquad \mathbf{g}^{\mathrm{T}} = \mathbf{p}^{\mathrm{T}} A^{-1}. \tag{3.9}$$

Izraz za $\mathbf{g}$ z desne množimo z A, nato obe strani transponiramo in upoštevamo simetričnost matrike A:

$$\mathbf{g}^{\mathrm{T}} A = \mathbf{p}^{\mathrm{T}}, \qquad A\mathbf{g} = \mathbf{p}. \tag{3.10}$$



Slika 3.2: Aproksimacija MLS funkcije $f(x) = \sin^2(5x) + x^2$ in nekatere bazne funkcije.

Odvajamo po i-ti neodvisni spremenljivki x_i:

$$A_{x_i}\mathbf{g} + A\mathbf{g}_{x_i} = \mathbf{p}_{x_i}, \qquad A\mathbf{g}_{x_i} = \mathbf{p}_{x_i} - A_{x_i}\mathbf{g}$$

Zdaj poznamo odvod $\mathbf{g}_{x_i}$, zato lahko odvajamo izraz za $\mathbf{\Phi}$ v (3.9):

$$\begin{aligned}\mathbf{\Phi}_{x_i} &= \mathbf{g}_{x_i}^{\mathrm{T}} B + \mathbf{g}^{\mathrm{T}} B_{x_i} \\ &= \left[A^{-1}(\mathbf{p}_{x_i} - A_{x_i}\mathbf{g})\right]^{\mathrm{T}} B + \mathbf{g}^{\mathrm{T}} B_{x_i} \\ &= (\mathbf{p}_{x_i}^{\mathrm{T}} - \mathbf{g}^{\mathrm{T}} A_{x_i}) A^{-1} B + \mathbf{g}^{\mathrm{T}} B_{x_i}. \end{aligned} \tag{3.11}$$

Odvode matrik A in B enostavno dobimo iz njihove definicije v (3.7). Produkt $A^{-1}B$ nastopa tako v baznih funkcijah kot v njihovih odvodih in ga lahko izračunamo le enkrat. S podobnim postopkom izrazimo tudi višje in mešane odvode baznih funkcij.

Neprimernost Householderjeve metode

Kot smo povedali zgoraj, je za reševanje problema najmanjših kvadratov načeloma boljša Householderjeva transformacija [Hea02], ki pravokotno matriko razstavi na faktorja:

$$P_{W(n\times m)} = Q_{(n\times n)} \begin{bmatrix} R_{(m\times m)} \\ 0_{((n-m)\times m)} \end{bmatrix},$$

kjer je R zgornjetrikotna matrika in je Q *ortogonalna*, kar pomeni, da velja $Q^{-1} = Q^{\mathrm{T}}$. S tema faktorjema lahko rešimo problem najmanjših kvadratov, pa tudi izrazimo *psevdoinverz* pravokotne matrike P_W^{-p} [Mey00]:

$$P_{W(m\times n)}^{-p} P_{W(n\times m)} = I_{(m\times m)}, \qquad P_W^{-p} = [\ R^{-1}\ 0^{\mathrm{T}}\ ]Q^{\mathrm{T}}.$$

Reševanje sistema (3.6) v smislu najmanjših kvadratov je algebrsko ekvivalentno operaciji:

$$\mathbf{a} = P_W^{-p} V \mathbf{u}_S,$$

zato lahko bazne funkcije izrazimo kot:

$$\phi_j(\mathbf{x}) = \sum_{i=1}^{m} p_i(\mathbf{x}) \left[P_W^{-p} V\right]_{i,j} = \mathbf{p}^{\mathrm{T}} P_W^{-p} V_{1\ldots n,j}. \tag{3.12}$$

Podobno kot pri uporabi metode normalnih enačb tudi tu dejansko ne računamo psevdoinverza, marveč rešimo problem najmanjših kvadratov s Householderjevo metodo.

Težave se žal pojavijo pri formulaciji odvodov. Na podoben način kot prej bi izrazili

$$\mathbf{\Phi}(\mathbf{x}) = \mathbf{g}^{\mathrm{T}}(\mathbf{x}) V(\mathbf{x}), \qquad \mathbf{g}^{\mathrm{T}} = \mathbf{p}^{\mathrm{T}} P_W^{-p}.$$

Množimo z desne s P_W in odvajamo po i-ti neodvisni spremenljivki x_i:

$$\mathbf{g}^{\mathrm{T}} P_W = \mathbf{p}^{\mathrm{T}}, \qquad \mathbf{g}_{x_i}^{\mathrm{T}} P_W + \mathbf{g}^{\mathrm{T}} P_{W,x_i} = \mathbf{p}_{x_i}^{\mathrm{T}}, \qquad \mathbf{g}_{x_i}^{\mathrm{T}} P_W = \mathbf{p}_{x_i}^{\mathrm{T}} - \mathbf{g}^{\mathrm{T}} P_{W,x_i}.$$

Tu se izvajanje ustavi, saj $P_W P_W^{-p} \neq I$, zato nima smisla z desne množiti s P_W^{-p}. Transponiranje ne pomaga, kajti na levi strani dobimo $P_W^{\mathrm{T}} \mathbf{g}_{x_i}$, česar zaradi ${P_W^{-p}}^{\mathrm{T}} P_W^{\mathrm{T}} \neq I$ ne moremo rešiti z množenjem s ${P_W^{-p}}^{\mathrm{T}}$ z leve. Na podobno oviro naletimo, če pri odvajanju (3.12) vpeljemo pomožni vektor na način: $\mathbf{\Phi}(\mathbf{x}) = \mathbf{p}^{\mathrm{T}}\mathbf{g}$. Pri metodi normalnih enačb teh problemov ni, ker je A kvadratna in velja $AA^{-1} = I$.
Sklep: Če potrebujemo odvode baznih funkcij, Householderjeva metoda ni uporabna.

Velikost nosilne domene

Premer nosilne domene $d_S(\mathbf{x})$ za MPM ne sme biti enak po celotni domeni Ω, marveč se mora prilagajati lokalni gostoti vozlišč [Liu03]. Slednja se spreminja, ker:

- vozlišča generiramo po enostavnem postopku, ki vsebuje naključnost,
- je vozlišča smiselno zgostiti tam, kjer pričakujemo bolj *divjo* rešitev, to je takšno z večjimi vrednostmi odvodov,
- želimo omogočiti *adaptivnost* MPM, ki naj avtomatično zgostijo vozlišča, kjer je to potrebno, in
- za točke blizu roba velik del nosilne domene pade izven domene Ω, kjer ni nobenega vozlišča.

Aproksimacija na sliki 3.2 tega ne upošteva, zato blizu desnega roba nosilna domena komaj vsebuje dovolj vozlišč in dobimo interpolacijo. Ob drugačni razporeditvi vozlišč bi imela lahko kaka točka le 2 nosilni vozlišči in matrika P_W ne bi imela več polnega ranga. Povečanje nosilne domene bi poslabšalo ujemanje na področjih z gostejšimi vozlišči. Nosilno domeno vsake točke $\mathbf{x}$ torej želimo določiti tako, da bo vsebovala primerno število vozlišč [FM04, Liu03].

V primeru pomanjkanja vozlišč je rang matrike P_W manjši od m, zato ne moremo določiti koeficientov za aproksimacijo. Vsa vozlišča ne prinesejo nove informacije – podobno kot pri polinomski interpolaciji nam aproksimacijo z monomi stopenj do 2 prepreči na primer postavitev vozlišč v dve vodoravni premici, ne glede na število vozlišč. Nosilno domeno moramo povečati in prej ali slej se bodo v njej pojavila vozlišča izven teh dveh premic, če le ni razporeditev vozlišč povsem izrojena.

Podobno nam zelo malo informacije prinesejo vozlišča, ki ležijo blizu roba nosilne domene. Njihova utež je namreč majhna, zato so vrednosti v pripadajoči vrstici P_W blizu nič. Če je rang matrike P_W, iz katere smo odstranili vrstico z majhno utežjo, enak $m-1$ in je torej rešljivost sistema odvisna od te skoraj ničelne vrstice, bo sistem zelo občutljiv. Iz obeh omenjenih razlogov je potrebno imeti pri velikosti nosilne domene precej rezerve.

Po drugi plati prevelika nosilna domena neke točke $\mathbf{x}$ pomeni močno povprečenje vozliščnih parametrov. Posledično bo poskusna funkcija v bližini $\mathbf{x}$ zelo mehka in zato ne bo mogla dobro opisati rešitve, če ta vsebuje ostre robove [Liu03].

Za lokalno določanje premera nosilne domene d_S lahko predpišemo [Liu03]:

$$d_S(\mathbf{x}) = \alpha_S \overline{d}(\mathbf{x}), \tag{3.13}$$

kjer je α_S *brezdimenzijska velikost* nosilne domene in $\overline{d}(\mathbf{x})$ *povprečna razdalja* med vozlišči v okolici točke $\mathbf{x}$. Slednja je mera za lokalno gostoto vozlišč in jo dobimo tako, da najprej izberemo *vzorčno domeno*, katere velikost označimo z $D_S(\mathbf{x})$. V izbrani vzorčni domeni preštejemo vozlišča in označimo njihovo število z n_{D_S}. Zdaj lahko izračunamo povprečno razdaljo med vozlišči v vzorčni domeni za eno-, dvo- in tridimenzionalni primer:

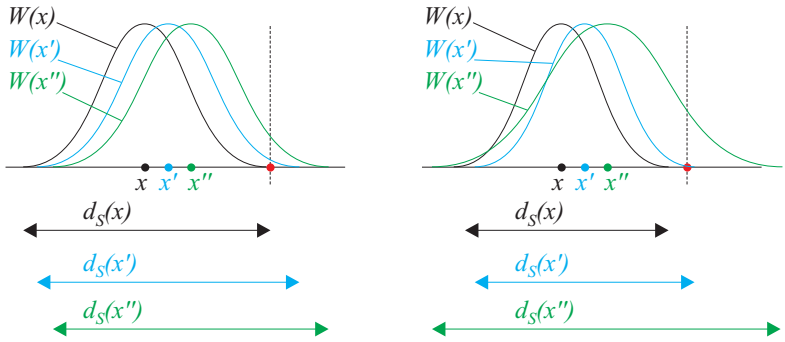
$$\overline{d}_{1D} = \frac{D_S}{n_{D_S}-1}, \qquad \overline{d}_{2D} = \frac{\sqrt{A_S}}{\sqrt{n_{D_S}}-1}, \qquad \overline{d}_{3D} = \frac{\sqrt[3]{V_S}}{\sqrt[3]{n_{D_S}}-1}, \tag{3.14}$$

kjer sta A_S in V_S površina oziroma volumen vzorčne domene [Liu03]. Z vstavljanjem dobljene povprečne razdalje $\overline{d}$ v (3.13) dobimo premer nosilne domene.

Velikost vzorčne domene mora biti takšna, da je povprečna razdalja med vozlišči v vzorčni domeni podobna povprečni razdalji v dobljeni nosilni domeni. Ustreznost izbire D_S lahko zatorej preverimo le naknadno s štetjem vozlišč v dobljeni nosilni domeni.

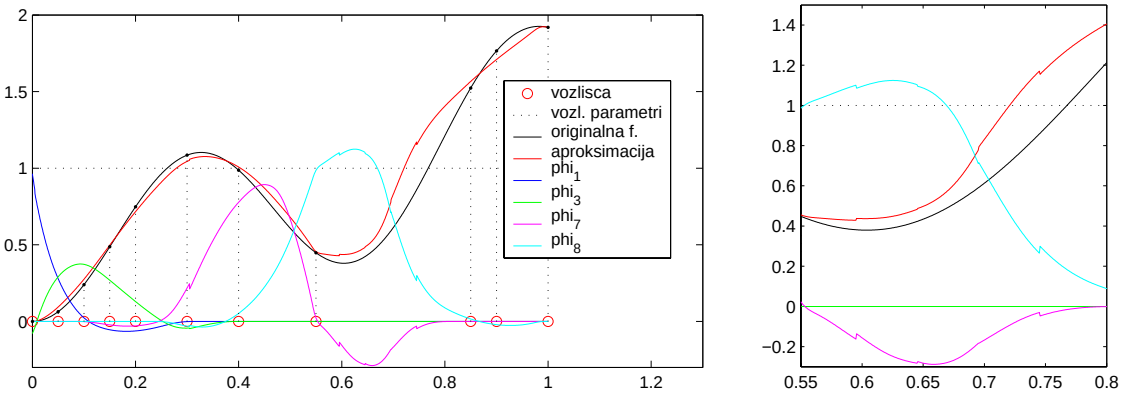
Zveznost aproksimacije MLS

V literaturi neomenjena težava takšnega določanja velikosti nosilne domene je, da se število vozlišč n_{D_S} vozlišč v vzorčni domeni spreminja diskretno, ko točka $\mathbf{x}$ potuje po domeni. S tem se nezvezno spremeni tudi končna velikost nosilne domene d_S. Tak primer prikazuje slika 3.3 – tri bližnje točke s pripadajočimi nosilnimi domenami in utežnimi funkcijami. Na levi sliki se premer nosilne domene spreminja zvezno, zato rdeče vozlišče zvezno vstopa v nosilno domeno (njegova utež se zvezno povečuje), ko se premikamo iz točke x proti x''. Zvezno vstopanje in izstopanje vozlišč zagotavlja tudi zveznost baznih in posledično poskusne funkcije. Na desni sliki se premer nosilne domene poveča nezvezno, zato se nezvezno poveča tudi utež rdečega vozlišča. V desnem primeru bosta bazna in poskusna funkcija nezvezni.



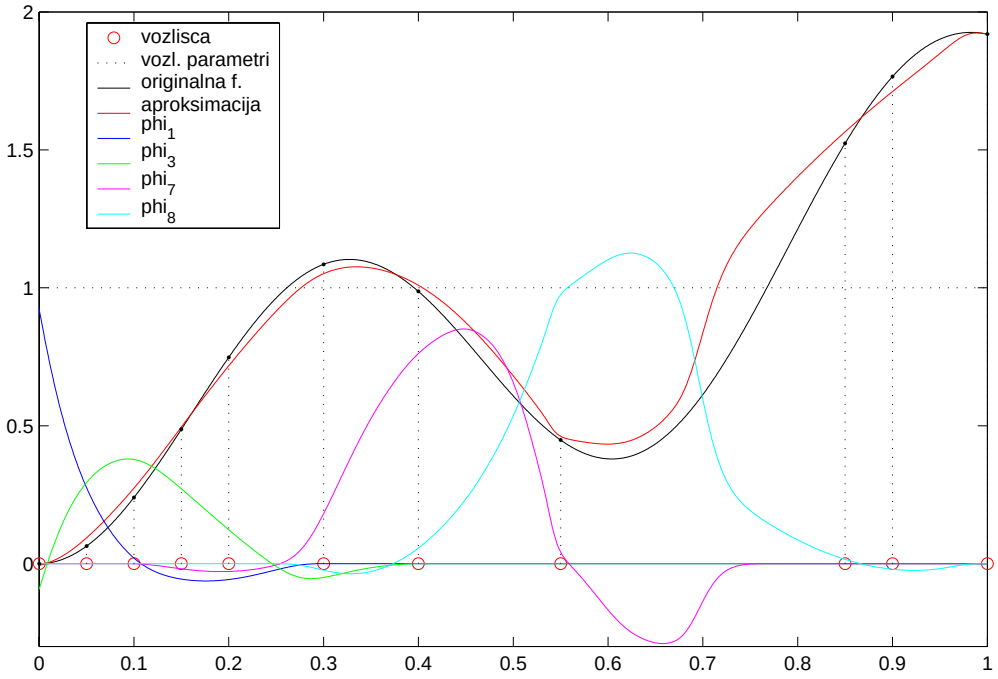
Slika 3.3: Bližnje točke x, x' in x'' s pripadajočimi nosilnimi domenami in utežnimi funkcijami.

Na sliki 3.4 je prikazana aproksimacija MLS, kjer smo velikost nosilne domene računali lokalno na prej opisan način. S primernejšim številom nosilnih vozlišč smo odpravili premočno prileganje blizu robov, vendar je aproksimacija nezvezna.



Slika 3.4: Primer aproksimacije z nezveznimi baznimi funkcijami MLS (levo) in detajl nezveznosti (desno).

Zveznost d_S lahko *vsilimo* tako, da vnaprej ovrednotimo povprečno razdaljo $\overline{d}$ na ekvidistantni mreži točk, vrednosti $\overline{d}$ v vmesnih točkah pa dobimo z interpolacijo, s čimer seveda nekoliko zgrešimo idealno velikost nosilne domene. V MPM tako spet uvedemo mrežo, ki pa je trivialna in ne prinese s seboj nobene izmed slabosti na mreži zasnovanih metod. Aproksimacijo na sliki 3.5 smo dobili na tak način. Interpolacija $\overline{d}$ ni opazno spremenila prileganja, čeprav je mreža za interpolacijo vsebovala le štiri točke $0, \frac{1}{3}, \frac{2}{3}$ in 1.
Sklep: Če želimo zvezne bazne funkcije MLS, se mora zvezno spreminjati velikost nosilne domene, kar lahko zagotovimo z interpolacijo povprečne razdalje.

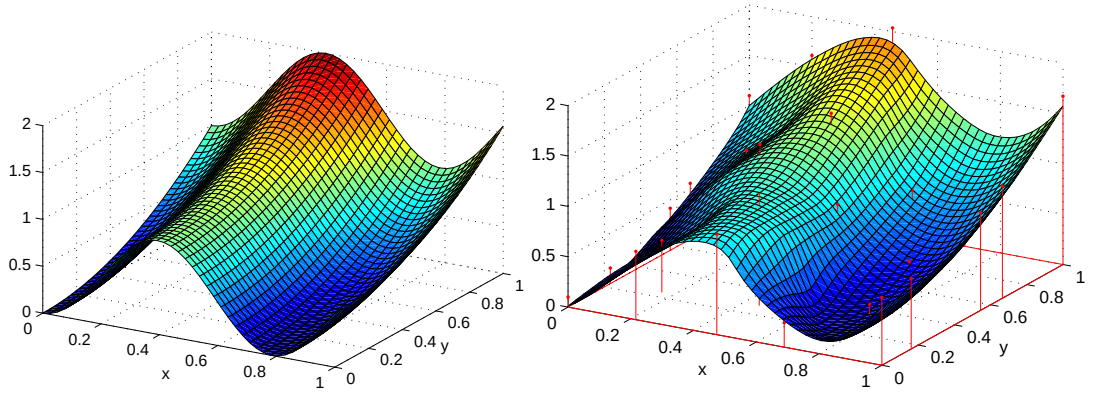


Slika 3.5: Aproksimacija s slike 3.4 z vsiljeno zveznostjo.

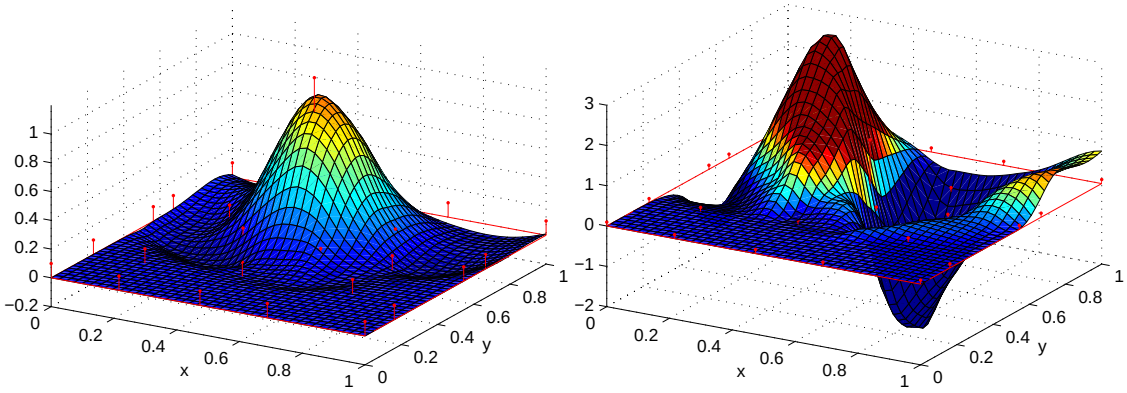
Primer dvodimenzionalne funkcije in pripadajoče zvezne dvodimenzionalne aproksimacije MLS prikazuje slika 3.6. Uporabili smo 25 vozlišč, vektor $\mathbf{p}$ je vseboval monome do 2. stopnje, velikost nosilne domene pa je bila $\alpha_S = 2.6$. Opazimo lahko, da gre za aproksimacijo, saj se istoležne vrednosti u in $\hat{u}$ razlikujejo. Pri tako majhnem številu vozlišč se oblika baznih funkcij pozna kot vdolbine in izbokline v aproksimaciji. Primer bazne funkcije in njenega parcialnega odvoda po x nahajamo na sliki 3.7.

Lastnosti baznih funkcij MLS

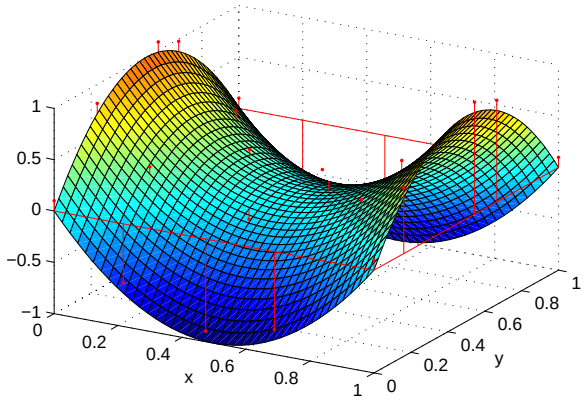
Bazne funkcije MLS se pomembno razlikujejo od tistih, dobljenih z interpolacijo [Liu03]. Kot smo videli, lahko poskrbimo za zveznost, kar pri interpolaciji v več kot eni dimenziji ni možno. Poleg tega bazne funkcije MLS nimajo Kroneckerjeve δ lastnosti, saj gre za aproksimacijo in ne interpolacijo vozliščnih parametrov. Pač pa so, podobno kot



Slika 3.6: Funkcija $u(x, y) = \sin^2(4 - 4x) + y^2$ (levo) in aproksimacija MLS na 25 vozliščih (desno). Vrednost aproksimacije je izračunana na ekvidistantni mreži (barvna ploskev), vozliščni parametri pa so prikazani kot rdeče palice s krogci na vrhu in so zaradi preglednosti narisani za 0.1 višje.



Slika 3.7: Ena izmed baznih funkcij (levo), uporabljenih za aproksimacijo na sliki 3.6, in njen parcialni odvod po x (desno).



Slika 3.8: Z baznimi funkcijami MLS drugega reda lahko točno predstavimo polinomsko funkcijo $u(x, y) = (2x - 1)^2 - (2y - 1)^2$ (levo).

interpolacijske, zmožne točno predstaviti vsako linearno kombinacijo funkcij, zastopanih v $\mathbf{p}$. Zato pravimo, da so reda konsistentnosti r, če $\mathbf{p}$ vsebuje vse monome reda $\leq r$.

Konsistentnost je ilustrirana na sliki 3.8. Vozliščni parametri ustrezajo vrednosti funkcije $u(x, y) = (2x - 1)^2 - (2y - 1)^2$, ki jo aproksimacija predstavi točno – vozliščni parametri bi se natanko dotikali površine, če ne bi bili narisani zamaknjeno, vdolbin in izboklin ni. Uporabljena so ista vozlišča in z njimi iste bazne funkcije kot na sliki 3.6.

3.2 Reševanje parcialnih diferencialnih enačb

Reševanje PDE z MPM poteka na podoben način kot pri MKE, razlike so le posledica drugačne konstrukcije baznih funkcij [Liu03]. Sistem navadnih enačb dobimo s kolokacijo ali uteženim ostankom, ki smo ju opisali v poglavju 2.1, ali na primer z variacijskim načelom. Morda ne bo odveč ponoviti enačbo, kakršne dobimo z uteženim ostankom:

$$\int_\Omega W_i(\mathbf{x}) r(\mathbf{x}) d\Omega = \int_\Omega W_i(\mathbf{x}) F(\mathbf{x}, \hat{u}, \ldots) d\Omega = 0, \tag{3.15}$$

kjer je število vseh testnih funkcij W_i enako številu neznank, to je vozliščnih parametrov $\mathbf{u}$. Pri reševanju s kolokacijo enačbe dobimo na enak način kot pri MKR in MKE. Po drugi plati pa uteženega ostanka ne moremo izraziti na enak način kot pri MKE, ki integral (3.15) izračuna kot vsoto integralov po posameznih elementih. Pri MPM namreč nimamo na voljo mreže elementov, ki bi predstavljala razdelitev domene Ω na disjunktne poddomene [Liu03].

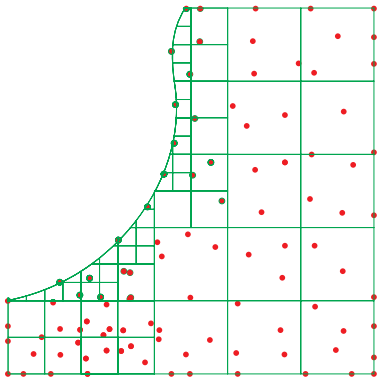
Pomagamo si lahko tako, da uvedemo pomožno *integracijsko mrežo* in integriramo podobno kot pri MKE. Lahko pa izberemo takšen nabor testnih funkcij, da je i-ta izmed njih različna od 0 le v neki majhni okolici i-tega notranjega vozlišča, ki jo imenujemo *integracijska* ali *kvadraturna domena* (angl. quadrature domain). Podobno smo z izborom utežne funkcije pri aproksimaciji MLS omejili nosilno domeno. V tem primeru za vsako posamezno enačbo obteženi ostanek integriramo lokalno, le znotraj pripadajoče kvadraturne domene, zato globalna mreža, ki bi pokrila celotno domeno, ni potrebna [AS02].

V nadaljevanju najprej predstavimo eno izmed metod, ki uporabljajo integracijsko mrežo, nato pa podrobno opišemo najbolj razširjeno metodo z lokalno integracijo. Na koncu podpoglavja pokažemo možne načine vsiljevanja robnih pogojev pri MPM.

3.2.1 Elementov prosta Galerkinova metoda

Elementov prosta Galerkinova metoda poskusno funkcijo konstruira z aproksimacijo MLS. Za potrebe integracije uvede pomožno integracijsko mrežo elementov in vrednost integrala (3.15) izračuna na vsakem elementu posebej [BLG94], zato morajo biti elementi enostavni

liki. Za oglišča elementov ni potrebno, da se ujemajo z vozlišči, uporabljenimi za poskusno funkcijo, zato je integracijsko mrežo enostavneje konstruirati kot mrežo za MKE. Vseeno mora biti integracijska mreža gostejša tam, kjer so gostejša vozlišča, ker so bazne funkcije tam bolj divje in jih je zato potrebno integrirati z manjšim razmikom med kvadraturnimi točkami. Metoda zahteva zvezne bazne funkcije.



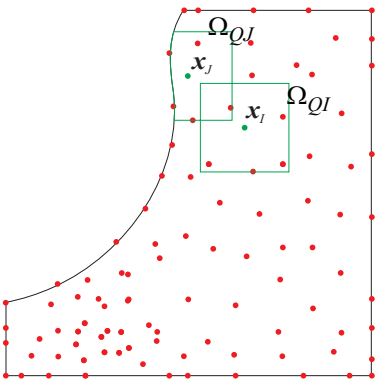
Slika 3.9: Razdelitev domene na integracijsko mrežo v EPG metodi.

Primer integracijske mreže prikazuje slika 3.9. Opazimo lahko težave ob robovih, kjer elementi niso pravilnih oblik. Ena od možnosti je, da jih nadalje drobimo toliko časa, da so prispevki nepravilnih elementov k skupnemu integralu majhni in zato velika relativna napaka tam ni problematična [Liu03]. Metodi EPG so sorodne metode, ki prav tako uporabljajo integracijsko mrežo, vendar namesto MLS uporabljajo druge tipe baznih funkcij [LAJ93, AS02].

3.2.2 Mreže prosta lokalna Petrov-Galerkinova metoda

Mreže prosta lokalna Petrov-Galerkinova metoda za konstrukcijo poskusne funkcije uporablja aproksimacijo MLS. Je predstavnica pravih MPM, ki razdelitve domene na mrežo ne potrebujejo niti za integracijo. Takšne metode so poleg MLPG še *meshless local boundary integral equation* (slov. mreže prosta enačba robnega integrala), ki je v bistvu poseben primer MLPG [AS02], in *interpolacijske metode* (angl. point interpolation methods, PIM), ki za razliko od MLPG uporabljajo interpolacijo bodisi s polinomskimi ali radialnimi baznimi funkcijami ali kombinacijo obojih. Nekatere izmed tovrstnih metod delujejo tudi z nezveznimi baznimi funkcijami, pri drugih pa je delovanje v takšnem primeru odvisno od tipa PDE [Liu03, LG03].

Osnovna ideja metode MLPG [AS02] je izbira po ene testne funkcije W_I za vsako notranje vozlišče $\mathbf{x}_I$, in sicer takšne, ki je od 0 različna le na majhni kvadraturni domeni v okolici $\mathbf{x}_I$, kot prikazuje slika 3.10. Na ta način za vsako notranje vozlišče dobimo po eno enačbo, v kateri seveda nastopajo tudi parametri ostalih vozlišč, podobno kot pri MKE. Ker baznih funkcij MLS ne moremo integrirati analitično, moramo odvisno od tipa te-



Slika 3.10: Kvadraturni domeni dveh vozlišč v metodi MLPG.

stnih funkcij po vsaki kvadraturni domeni Ω_{QI} razmestiti *kvadraturne točke* za numerično integracijo. V enačbi vozlišča $\mathbf{x}_I$ zato nastopajo parametri vseh vozlišč, ki ležijo znotraj nosilne domene katerekoli kvadraturne točke v Ω_{QI} [Liu03].

V družini metod MLPG lahko uporabimo različne testne funkcije, poleg tega pa lahko pri integraciji izrek o divergenci uporabimo ali pa ne. V [AS02] najdemo sistematično razdelitev različic MLPG1 do MLPG6 glede na ti dve lastnosti in podrobno analizo njihove natančnosti. Najbolje se obnesejo MLPG1, MLPG2 in MLPG5, ki si jih bomo podrobneje ogledali.

MLPG1

Prva različica za testne funkcije uporablja utežne funkcije v obliki klobuka z vrhom v vozlišču. To je lahko bodisi ista utežna funkcija, kot smo jo uporabili pri konstrukciji baznih funkcij MLS [LA00], ali pa kakšna enostavnejša, na primer v dveh dimenzijah [Liu03]:

$$W_I(x,y) = \left(1 - \left(\frac{x - x_I}{d_Q/2}\right)^2\right) \cdot \left(1 - \left(\frac{y - y_I}{d_Q/2}\right)^2\right), \tag{3.16}$$

kjer je kvadraturna domena kvadratna in je d_Q dolžina njene stranice. V vsakem primeru izbira utežne funkcije določa obliko kvadraturne domene. Velikost kvadraturne domene določimo podobno kot velikost nosilne domene:

$$d_Q(\mathbf{x}) = \alpha_Q \overline{d}(\mathbf{x}),$$

kjer je α_Q njena brezdimenzijska velikost.

Metoda MLPG1 izrek o divergenci uporabi enkrat, zato morajo biti bazne funkcije za reševanje PDE reda r vsaj reda konsistentnosti $r - 1$. Testne funkcije morajo biti enkrat zvezno odvedljive [AS02].

Oglejmo si, kako prenašanje odvodov z baznih na testne funkcije vpliva na izvedbo metode MLPG1. Denimo, da v PDE in s tem v izrazu za uteženi ostanek nastopa odvod v_{x_i}, kjer je v bodisi u ali kakšen njen odvod, lahko tudi višji ali mešan – drugače povedano, v_{x_i} je u, odvajana vsaj enkrat po x_i in nič- ali večkrat po vsaki izmed ostalih neodvisnih spremenljivk. Potem izrek o divergenci (2.6) pravi:

$$\int_{\Omega_{Qi}} W_i v_{x_i} d\Omega_{Qi} = \int_{\Gamma_{Qi}} W_i v \, n_i \, d\Gamma_{Qi} - \int_{\Omega_{Qi}} W_{i,x_i} v \, d\Omega_{Qi}, \tag{3.17}$$

kjer je Γ_{Qi} rob kvadraturne domene Ω_{Qi}. Če kvadraturna domena ne seka roba globalne domene ($\Omega_{Qi} \cap \Gamma = \emptyset$), prvi člen odpade, ker je $W_i(\mathbf{x}) = 0$, če $\mathbf{x} \in \Gamma_{Qi}$. V nasprotnem primeru, kakršen je na primer kvadraturna domena Ω_{QJ} na sliki 3.10, lahko bodisi izračunamo tudi robni integral v prvem členu ali pa testne funkcije definiramo tako, da so na robu globalne domene vedno enake 0 [AS02].

MLPG2

Za testne funkcije lahko uporabimo Diracove δ funkcije z vrhovi v vozliščih, kar metodo uteženega ostanka spremeni v kolokacijo [AS02]. Integral se poenostavi v ovrednotenje integranda v vozlišču, izreka o divergenci pa ne moremo uporabiti, ker testne funkcije niso odvedljive. Bazne funkcije morajo biti zato istega reda konsistentnosti kot diferencialna enačba. Kot večina kolokacijskih metod ima tudi MLPG2 manjšo računsko zahtevnost na račun slabše natančnosti [FM04, AS02].

MLPG5

Zanimiv pristop je uporaba testnih funkcij oblike:

$$W_i(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in \Omega_{Qi} \\ 0 & \mathbf{x} \notin \Omega_{Qi}, \end{cases}$$

kjer so kvadraturne domene definirane tako, da nikjer ne segajo čez rob globalne domene Ω. Z uporabo izreka o divergenci spet dobimo (3.17). Zdaj odpade drugi člen, saj je odvod testne funkcije povsod v kvadraturni domeni enak 0. Integracija je potrebna le po robu posamezne kvadraturne domene, kar metodo močno poenostavi. Bazne funkcije za metodo MLPG5 morajo biti enakega reda konsistentnosti kot pri MLPG1 [AS02].

MLPG5 ima težave s PDE, v katerih nastopa funkcija u sama ali odvajana le po času. Takšen primer je difuzijska enačba s časovnim odvodom u_t. Če namreč na tem členu uporabimo izrek o divergenci, dobimo v prvem členu na desni strani (3.17) nedoločen integral $\int u dx$ oziroma izraze oblike $u_i \int \phi_i dx$, ki jih ni možno izračunati numerično, za analitično integracijo pa so bazne funkcije običajno preveč zapletene. Če izreka o divergenci na takšnih členih PDE ne uporabimo, integrali po notranjosti kvadraturnih domen ostanejo, s čimer metoda MLPG5 izgubi prednost pred MLPG1.

3.2.3 Vsiljevanje robnih pogojev

Pomembna razlika med baznimi funkcijami MKE in aproksimacijskimi baznimi funkcijami MLS je, da slednje nimajo Kroneckerjeve δ lastnosti [Liu03]. Za vsiljevanje Dirichletovih robnih pogojev zato ne zadošča enak postopek kot pri MKE, kjer parametre robnih vozlišč kar izenačimo z robnimi pogoji. V primeru Neumannovih ali drugih vrst robnih pogojev pa nam tako in tako niti Kroneckerjeva δ lastnost ne pomaga in moramo v vsakem primeru uporabiti enega od spodaj opisanih postopkov. V splošnem primeru imamo dan nabor k *omejitev*:

$$C_k(\mathbf{x}) = 0,\ \mathbf{x} \in \Theta_k,\ \Theta_k \subseteq \Omega,$$

kjer je Θ_k bodisi točka, krivulja, površina ali telo [Liu03]. Če je C_k robni pogoj, velja $\Theta_k \subseteq \Gamma$. Omejili se bomo na *linearne omejitve* – C_k naj bo linearna funkcija $\hat{u}$ in njenih odvodov, kajti v splošnem primeru končni sistem ne bi bil linearen. V omejitvah smejo nastopati odvodi največ takšnega reda, kot je red konsistentnosti baznih funkcij. Najbolje bi bilo bazne funkcije konstruirati tako, da bi $\hat{u}$ že po definiciji zadoščala omejitvam, kar pa pri MPM ni možno, saj so bazne funkcije povsem določene s položaji vozlišč [Liu03].

Kazenska metoda in Lagrangeovi multiplikatorji

Omejitve lahko vsilimo tako, da namesto ostanka r minimiziramo spremenjen ostanek $\tilde{r}$:

$$\tilde{r}(\mathbf{x}) = r(\mathbf{x}) + \sum_k \alpha C_k^p(\mathbf{x}),\ C_k^p(\mathbf{x}) = \begin{cases} C_k(\mathbf{x}) & \mathbf{x} \in \Theta_k \\ 0 & \text{sicer.} \end{cases}$$

Postopku pravimo *kazenska metoda* (angl. penalty method), konstanti α pa *kazenski faktor*, ki mora biti pravilno izbran. Teoretično bi izbira $\alpha = \infty$ točno vsilila omejitve, vendar pri numeričnem računanju neskončnosti ne moremo uporabiti, marveč moramo najti kompromis. Posledica premajhnega kazenskega faktorja bodo le približno izpolnjeni robni pogoji, v primeru prevelikega pa bo zaradi omejene natančnosti računanja trpela natančnost rešitve v notranjosti domene [Liu03].

Dilemi o izbiri kazenskega faktorja se lahko izognemo tako, da ga nadomestimo z *Lagrangeovimi multiplikatorji* (angl. Lagrange multipliers) λ_k, ki so funkcije $\mathbf{x}$ in jih definiramo s posebnimi vozlišči in enostavnejšimi baznimi funkcijami. Na Lagrangeove multiplikatorje lahko gledamo kot na *pametne sile*, ki prisilijo rešitev k izpolnjevanju omejitev [Liu03]. Slabost te metode je zapletenost in večja računska zahtevnost. Za vsako bazno funkcijo, uporabljeno za konstrukcijo kakega Lagrangeovega multiplikatorja, namreč končni sistem navadnih enačb naraste za eno enačbo [AS02]. Kazenska metoda in Lagrangeovi multiplikatorji so uporabni tako za EPG kot MLPG metode.

Kolokacija v robnih vozliščih in transformacijska metoda

Dasiravno zaradi odsotnosti Kroneckerjeve δ lastnosti vrednost poskusne funkcije, konstruirane z aproksimacijo MLS, v vozlišču x_I ni enaka vozliščnemu parametru u_I, marveč je tako kot v vsaki drugi točki linearna kombinacija baznih funkcij, jo lahko enostavno izenačimo z zahtevanim Dirichletovim robnim pogojem $\overline{u}$:

$$\hat{u}(\mathbf{x}_I) = \sum_j \phi_j(\mathbf{x}_I) u_j = \overline{u}(\mathbf{x}_I). \tag{3.18}$$

Dobljena enačba vsili Dirichletov robni pogoj v vozlišču x_I, podobne enačbe pa lahko nastavimo za druge oblike omejitev. Med vozlišči robni pogoji niso izpolnjeni točno, kar ne predstavlja težav, če so vozlišča dovolj gosta [AS02].

S postopkom smo pridobili eno enačbo za vsako robno vozlišče. V metodah MLPG, ki globalni sistem navadnih enačb sestavijo iz po ene enačbe za vsako vozlišče, nove enačbe skupaj s tistimi oblike (3.15) za notranja vozlišča tvorijo sistem natanko takšne velikosti, kot je število neznank. Pri EPG metodi robnih pogojev ne moremo vsiljevati s kolokacijo v robnih vozliščih [Liu03].

Namesto uporabe enačb oblike (3.18) v končnem sistemu navadnih enačb jih lahko zberemo posebej in z njimi formuliramo transformacijo R vektorja baznih funkcij: $\mathbf{\Psi} = \mathbf{\Phi} R$, pri čemer imajo nove bazne funkcije $\mathbf{\Psi}$ Kroneckerjevo δ lastnost v robnih vozliščih, in z R pomnožimo tudi togostno matriko [AS02]. Dobljena *transformacijska metoda* je algebrsko ekvivalentna kolokaciji v robnih vozliščih.

3.3 Reševanje dvodimenzijske difuzijske enačbe

Za reševanje difuzijske enačbe smo izbrali metodo MLPG1, ker je značilna predstavnica MPM in je tudi v literaturi priporočena za tovrstne probleme [LA00, AS02, Liu03].

3.3.1 Enačba in poskusna funkcija

Izpeljava linearnega sistema z metodo MLPG1 za difuzijsko enačbo:

$$u_t - c\,(u_{xx} + u_{yy}) - g = 0 \tag{3.19}$$

na domeni Ω z robom Γ z začetnimi in Dirichletovimi robnimi pogoji:

$$u(x, y, t) = \overline{u}, \text{ če } (x, y) \in \Gamma, \qquad u(x, y, 0) = u_0$$

je podobna izpeljavi za MKE v razdelku 2.5.5. Najprej definiramo poskusno funkcijo in njene odvode po času in prostoru:

$$\hat{u}(x, y, t) = \sum_{\mathbf{x}_j \in \Omega_S(x,y)} u_j(t) \phi_j(x, y),$$

$$\hat{u}_t = \sum_{\mathbf{x}_j \in \Omega_S(x,y)} u_j'(t)\phi_j(x,y), \qquad \hat{u}_x = \sum_{\mathbf{x}_j \in \Omega_S(x,y)} u_j(t)\phi_{j,x}(x,y),$$

kjer je $\Omega_S(x,y)$ nosilna domena točke (x,y), $u_j(t)$ pa je parameter vozlišča j v času t. Podobno izrazimo tudi $\hat{u}_y$.

3.3.2 Diskretizacija prostora

PDE bomo najprej pretvorili v sistem NDE. Za notranja vozlišča sestavimo enačbe, ki zahtevajo, da so integrali ostanka, uteženega s testnimi funkcijami, enaki nič:

$$\int_{\Omega_{Qi}} W_i \left[\hat{u}_t - c\left(\hat{u}_{xx} + \hat{u}_{yy}\right) - g\right] d\Omega_{Qi} = 0.$$

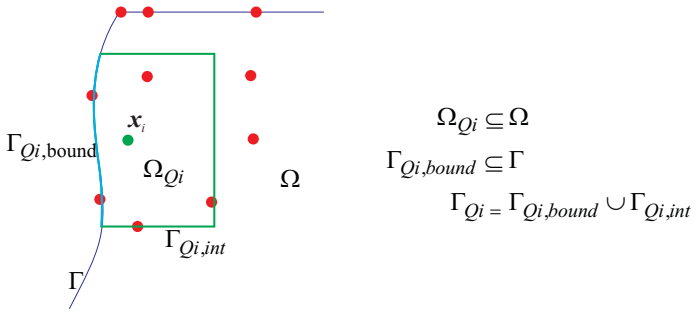
Integral najprej razstavimo na posamezne člene:

$$\int_{\Omega_{Qi}} W_i \hat{u}_t d\Omega_{Qi} - c\int_{\Omega_{Qi}} W_i\left(\hat{u}_{xx} + \hat{u}_{yy}\right) d\Omega_{Qi} - \int_{\Omega_{Qi}} W_i g d\Omega_{Qi} = 0, \tag{3.20}$$

nato pa se drugega odvoda $\hat{u}_{xx}$ znebimo z uporabo izreka o divergenci (3.17):

$$\int_{\Omega_{Qi}} W_i \hat{u}_{xx} d\Omega_{Qi} = \int_{\Gamma_{Qi}} W_i \hat{u}_x n_x d\Gamma_{Qi} - \int_{\Omega_{Qi}} W_{i,x}\hat{u}_x d\Omega_{Qi}.$$

Rob Γ_{Qi} kvadraturne domene je sestavljen iz notranjega dela $\Gamma_{Qi,int}$ in robnega dela, ki se pokriva z robom Γ domene: $\Gamma_{Qi,bound} = \Omega_{Qi} \cap \Gamma$. Odnosi med vsemi deli domene so skicirani na sliki 3.11. Vrednost testne funkcije na $\Gamma_{Qi,int}$ je enaka 0, zato lahko integral po celotnem robu Γ_{Qi} kvadraturne domene nadomestimo z integralom po robnem delu $\Gamma_{Qi,bound}$.



Slika 3.11: Odnos med kvadraturno domeno Ω_{Qi}, njenim robom Γ_{Qi} in globalno domeno Ω.

Podobno storimo z drugim odvodom $\hat{u}_{yy}$, s čimer iz (3.20) dobimo:

$$\begin{aligned}\int_{\Omega_{Qi}} W_i \hat{u}_t d\Omega_{Qi} - c \int_{\Gamma_{Qi,bound}} W_i \left(\hat{u}_x n_x + \hat{u}_y n_y\right) d\Gamma_{Qi,bound} \\ + c \int_{\Omega_{Qi}} \left(W_{i,x}\hat{u}_x + W_{i,y}\hat{u}_y\right) d\Omega_{Qi} \\ - \int_{\Omega_{Qi}} W_i g d\Omega_{Qi} \qquad\qquad = 0.\end{aligned}$$

Enačbo moramo izraziti z vozliščnimi parametri in baznimi funkcijami, da bomo sistem lahko zapisali v matrični obliki, zato vstavimo izraze za $\hat{u}$ in njene odvode. Hkrati še vse integrale zapišemo kot ustrezne enojne in dvojne integrale:

$$\begin{aligned}\iint_{\Omega_{Qi}} W_i(x,y) \sum_{\mathbf{x}_j \in \Omega_S(x,y)} u_j'(t)\phi_j(x,y)dxdy \\ -c\int_{\Gamma_{Qi,bound}} W_i(x,y) \sum_{\mathbf{x}_j \in \Omega_S(x,y)} u_j(t)\left[\phi_{j,x}(x,y)n_x(x,y) + \phi_{j,y}(x,y)n_y(x,y)\right] ds \\ +c\iint_{\Omega_{Qi}} \sum_{\mathbf{x}_j \in \Omega_S(x,y)} u_j(t)\left[W_{i,x}(x,y)\phi_{j,x}(x,y) + W_{i,y}(x,y)\phi_{j,y}(x,y)\right] dxdy \\ - \iint_{\Omega_{Qi}} W_i(x,y)g(x,y,t)dxdy = 0.\end{aligned}$$

Od tu naprej bomo zaradi kompaktnosti zapisa izpuščali neodvisne spremenljivke. Neznanke u_j in zaradi njih tudi znake za vsote moramo prenesti iz integralov:

$$\begin{aligned}\sum_{\mathbf{x}_j \in \Omega_{SQi}} u_j' \iint_{\Omega_{Qi}} W_i\phi_j dxdy \\ -c \sum_{\mathbf{x}_j \in \Omega_{SQi}} u_j \int_{\Gamma_{Qi,bound}} W_i\left[\phi_{j,x}n_x + \phi_{j,y}n_y\right] ds \\ +c \sum_{\mathbf{x}_j \in \Omega_{SQi}} u_j \iint_{\Omega_{Qi}} \left[W_{i,x}\phi_{j,x} + W_{i,y}\phi_{j,y}\right] dxdy \\ - \iint_{\Omega_{Qi}} W_i g dxdy = 0,\end{aligned} \tag{3.21}$$

kjer je Ω_{SQi} *nosilna domena kvadraturne domene* Ω_{Qi}:

$$\Omega_{SQi} = \bigcup_{(x,y)\in\Omega_{Qi}} \Omega_S(x,y).$$

V skladu s tradicionalnim poimenovanjem označimo s C^{int} koeficiente, ki v (3.21) nastopajo pomnoženi s časovnimi odvodi vozliščnih parametrov, s K^{int} tiste ob parametrih samih in s $\mathbf{f}^{int}$ člene, ki nastopajo samostojno. Z nadpisom ^{int} smo označili, da gre za

tiste vrstice matrik, ki ustrezajo notranjim vozliščem.

$$\begin{aligned} K_{i,j}^{int} =& c\left[\iint_{\Omega_{Qi}} \left[W_{i,x}\phi_{j,x} + W_{i,y}\phi_{j,y}\right] dxdy - \int_{\Gamma_{Qi,bound}} W_i \left[\phi_{j,x}n_x + \phi_{j,y}n_y\right] ds\right], \\ C_{i,j}^{int} =& \iint_{\Omega_{Qi}} W_i\phi_j dxdy, \\ f_i^{int} =& \iint_{\Omega_{Qi}} W_i g dxdy. \end{aligned} \tag{3.22}$$

Robne pogoje vsilimo s kolokacijo v robnih vozliščih po enačbi 3.18. Vse enačbe za robna in notranja vozlišča lahko združimo v sistem NDE:

$$C\mathbf{u}'(t) + K\mathbf{u}(t) - \mathbf{f}(t) = 0, \tag{3.23}$$

kjer so celotne matrike:

$$K_{i,j} = \begin{cases} \phi_j(\mathbf{x}_i) & \mathbf{x}_i \in \Gamma \text{ in } \mathbf{x}_j \in \Omega_{Si} \\ K_{i,j}^{int} & \mathbf{x}_i \notin \Gamma \text{ in } \mathbf{x}_j \in \Omega_{SQi} \\ 0 & \text{sicer,} \end{cases}$$

$$C_{i,j} = \begin{cases} C_{i,j}^{int} & \mathbf{x}_i \notin \Gamma \text{ in } \mathbf{x}_j \in \Omega_{SQi} \\ 0 & \text{sicer,} \end{cases}$$

$$f_i = \begin{cases} \overline{u}(\mathbf{x}_i) & \mathbf{x}_i \in \Gamma, \\ f_i^{int} & \text{sicer.} \end{cases}$$

Primere, ko so v matrikah K in C ničle, smo zapisali eksplicitno, da je razvidna redkost obeh matrik – vrednosti 0 sicer sledijo že iz integralov v K^{int} in C^{int}. Pač pa matriki nista simetrični. Že K^{int} ni simetrična, ker so testne funkcije drugačne od baznih. Poleg tega vsiljevanje robnih pogojev s kolokacijo povzroči posebno obliko robnih enačb in s tem nekaterih vrstic, ne pa tudi istoležnih stolpcev matrike. Ker enačbe za parametre robnih vozlišč niso trivialne, jih tudi ne moremo odstraniti iz sistema in preštevilčiti ostalih vozlišč kot pri MKE.

3.3.3 Diskretizacija časa

Opazimo lahko, da imata sistema NDE, dobljena z MKE (2.21) in z MLPG1 (3.23) enako obliko, zato tudi v tem primeru Crank-Nicolsonova shema da končni linearni sistem:

$$(2C + \Delta t \cdot K)\mathbf{u}^{(k+1)} = (2C - \Delta t \cdot K)\mathbf{u}^{(k)} + 2\Delta t \cdot \mathbf{f}(t + \frac{\Delta t}{2}).$$

Sistem je težji za reševanje od tistega, dobljenega z MKE, ker:

- matrike niso simetrične,
- je sistem nekoliko večji, kajti vsebuje tudi enačbe robnih vozlišč, in
- so matrike gostejše, ker vsaka nosilna domena Ω_{SQi} tipično vsebuje več vozlišč kot je stopnja sosednosti vozlišč v mreži MKE.

Poglavje 4

Natančnost mreže proste metode

V tem poglavju najprej preverimo metode končnih razlik, končnih elementov in mreže proste metode z umetnim testom. Opišemo dva testna primera, na katerih bomo metode preizkušali in primerjali, ter za boljšo predstavo navedemo rešitve obeh primerov z MKE in MKR. Nato naštejemo parametre MPM in utemeljimo izbiro nekaterih izmed njih, za ostale pa podamo intervale smiselnih vrednosti. Vpliv dveh najpomembnejših parametrov, števila vozlišč in stopnje baznih funkcij, neformalno ilustriramo tako, da z MPM nekajkrat rešimo oba testna primera. V naslednjem razdelku navedemo optimalne vrednosti vseh parametrov in natančno opišemo poskuse, s pomočjo katerih smo te vrednosti našli. Na koncu poglavja neposredno primerjamo MKR, MKE in MPM glede na to, kako natančno zmorejo rešiti oba testna primera, in povzamemo vse ugotovitve.

4.1 Vrednotenje napak

Vse metode, za katere smo v predhodnih poglavjih opisali reševanje difuzijske enačbe, smo preverili na treh testnih primerih. Dobljeno numerično rešitev smo ovrednotili na pravilni mreži in od nje odšteli vrednost analitične rešitve, s čimer smo dobili absolutno napako. Relativna napaka v našem primeru ni uporabna, ker je točna vrednost rešitve na robovih nekaterih testnih primerov enaka 0. Za vizualno preverjanje bomo na grafih vedno prikazovali rešitev in absolutno napako, ovrednoteni na pravilni mreži 52×52 točk.

Za kvantitativno primerjavo metod smo absolutne napake, ovrednotene na gostejši mreži 102×102 točki, zložili v vektor ter za skalarni meri napake izbrali neskončno in evklidsko normo dobljenega vektorja. Obe skalarni meri sta bili v poskusih močno korelirani – metoda z majhno neskončno normo je imela večinoma tudi majhno evklidsko normo in obratno, zato bomo v nadaljevanju uporabljali le neskončno normo, to je največjo absolutno napako.

4.2 Testni primeri

Izbrali smo tri testne primere: umetnega, ki preverja pravilnost izvedbe metod, ter dva realnejša, ki sta se razlikovala v obliki domene.

4.2.1 Umetni primer – stacionarno stanje

Vse tri izbrane metode smo najprej preverili na enotskem kvadratu ob naslednjih začetnih pogojih:

$$\begin{aligned} u_1(x,y) &= 0.5 + 0.67x - 1.67y, \\ u_2(x,y) &= 3(x-0.5)^2 - 3(y-0.5)^2, \\ u_3(x,y) &= 0.0067e^{5x}\sin(5y). \end{aligned}$$

Robni pogoji so bili vsakič enaki začetnim na robovih, konstante pa izbrane tako, da so največje absolutne vrednosti u okrog 1. Vsem trem funkcijam je skupno, da predstavljajo *stacionarno stanje* sistema, ker velja $\nabla^2 u_i = 0$. S točno metodo za reševanje PDE se rešitev skozi čas ne bi spreminjala. Za približno metodo to ne velja, vsekakor pa mora tudi ta konvergirati v stanje, podobno začetnim pogojem [Liu03].

Neskončno normo razlike med začetnim in končnim stanjem povzema tabela 4.1. *MPM1* označuje MPM tipa MLPG1 z aproksimacijo MLS z monomi stopenj do 1, *MPM2* pa z monomi stopenj do 2. Za MKR smo uporabili mrežo 100 točk, za MKE in MPM pa 100 vozlišč. MKR odvode u aproksimira, kot bi bila u polinom 2. stopnje, zato je razlika 0 natanko v tem primeru, pri nepolinomski funkciji ali polinomu višjih stopenj pa je razlika majhna, vendar večja od 0. Uporabljena MKE aproksimira u linearno, zato je razlika 0 le pri linearni funkciji. Pri MPM z redom konsistentnosti r je razlika 0 tedaj, če je u polinom stopnje največ r. Razlike so v vseh primerih majhne, zato sklepamo, da metode v teh treh primerih delujejo pravilno.

Tabela 4.1: Razlike med začetnim in končnim stanjem pri stacionarnih začetnih pogojih.

zač. pogoji	MKR	MKE	MPM1	MPM2
u_1	0	0	0	0
u_2	0	0.007	0.012	0
u_3	0.009	0.010	0.022	0.008

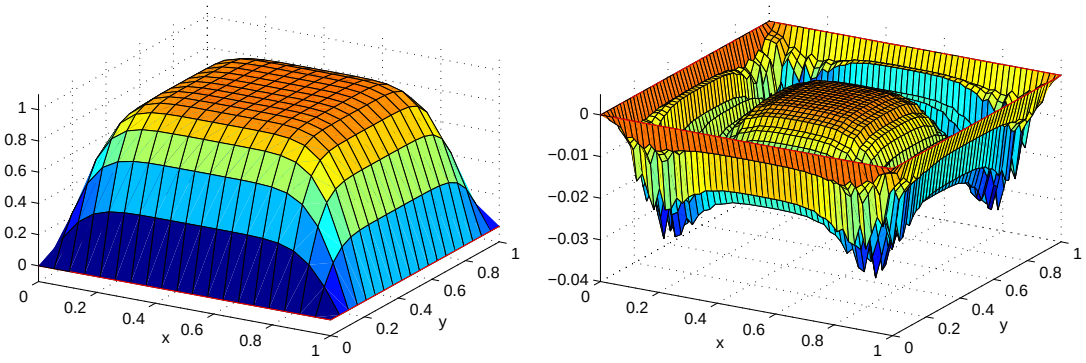
4.2.2 Testni primer s pravilno obliko domene

Testni primer z domeno v obliki enotskega kvadrata smo opisali že pri predstavitvi difuzijske enačbe v razdelku 2.2.2. Numerične rešitve smo primerjali z analitično rešitvijo [CJ59] ob času $t = 5$, ki se od začetnega stanja razlikuje dovolj, da lahko ugotavljamo

pravilnost časovnega razvoja rešitve in s tem pravilnost delovanja metode. Hkrati je dovolj blizu začetnemu stanju, da se v njej pozna odsekanost začetnih pogojev, zato se lahko dobro obnese le metoda, ki zna predstaviti razmeroma divjo rešitev.

Rešitev z metodo končnih razlik

MKR smo *prototipno* implementirali v programskem okolju Matlab [Matb]. Levi del slike 4.1 prikazuje rešitev testnega primera s pravilno obliko domene, izračunano na mreži $20 \times 20 = 400$ točk. Rešitev je prikazana na enaki gostoti mreže, kot je bila uporabljena za izračun. Absolutno napako (slika 4.1 desno) smo dobili z bilinearno interpolacijo rešitve na mrežo 52×52 točk in odštevanjem analitične rešitve.

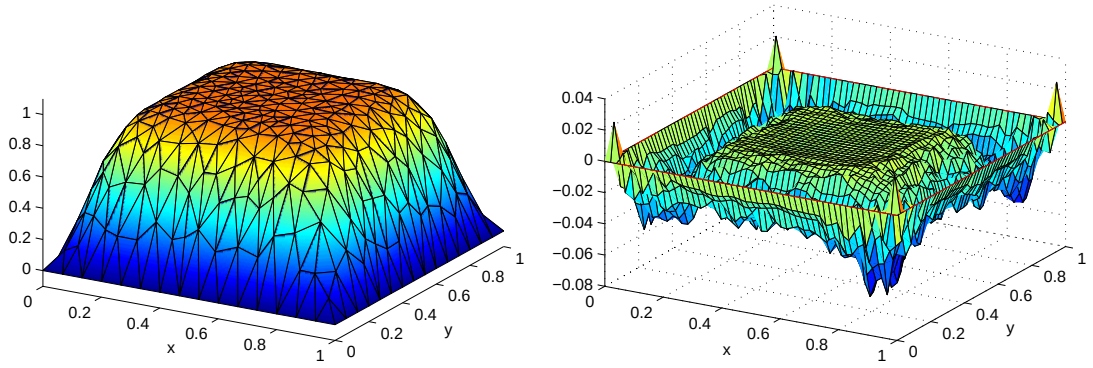


Slika 4.1: Rešitev testnega primera s pravilno obliko domene z MKR na mreži $20 \times 20 = 400$ točk (levo) in napaka (desno).

Rešitev z metodo končnih elementov

Za reševanje difuzijske enačbe z MKE smo uporabili programski paket Femlab[1], ki uporablja trikotne elemente in bazne funkcije prvega reda [Com]. Femlab je splošno priznan kot referenca na področju MKE [Sch03, Ver04]. Mrežo elementov smo generirali z vgrajenim algoritmom s privzetimi parametri, za časovno integracijo pa smo uporabili metodo `ode23s` [Matb], ki temelji na spremenjeni *Rosenbrockovi formuli* drugega reda [SR97]. Absolutno in relativno toleranco smo nastavili na 10^{-5}. Slika 4.2 prikazuje rešitev primera s pravilno obliko domene, izračunano na mreži s 740 elementi in 401 vozliščem (levo), in napako (desno).

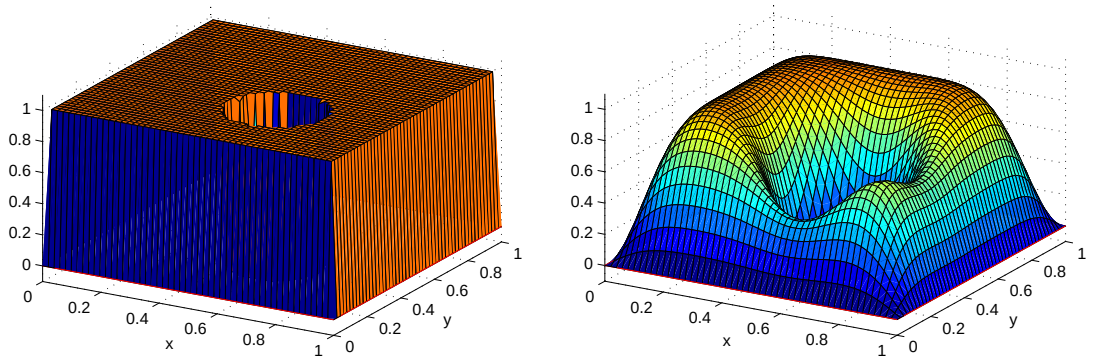
[1] Podjetje Comsol, avtor paketa Femlab, je v novejših različicah le-tega preimenovalo v Comsol Multiphysics.



Slika 4.2: Rešitev testnega primera s pravilno obliko domene z MKE na mreži s 740 elementi in 401 vozliščem (levo) in napaka (desno).

4.2.3 Testni primer z nepravilno obliko domene

Drugi primer je podoben, le da smo v domeno izvrtali luknjo s premerom 0.3 in središčem v točki $(0.6, 0.35)$, na robu luknje pa smo enako kot na zunanjem robu kvadrata predpisali robni pogoj $\overline{u} = 0$. Ker analitične rešitve tega primera ne poznamo, smo se morali zadovoljiti z referenčno rešitvijo, dobljeno z MKE s 50 000 elementi, 25 378 vozlišči in skrajšanim časovnim korakom $\Delta t = 0.02$. S tako velikim številom vozlišč smo zagotovili, da je referenčna rešitev vsaj za en red velikosti natančnejša od rešitev, ki jih bomo z njo primerjali. Na sliki 4.3 so prikazani začetni pogoji za primer z luknjo (levo) in referenčna rešitev ob času $t = 5$ (desno).

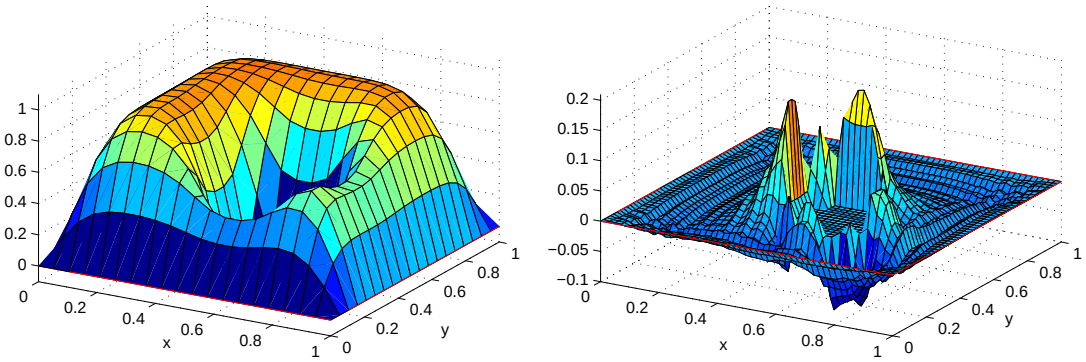


Slika 4.3: Začetni pogoji za testni primer z luknjo (levo) in referenčna rešitev ob času $t = 5$ (desno).

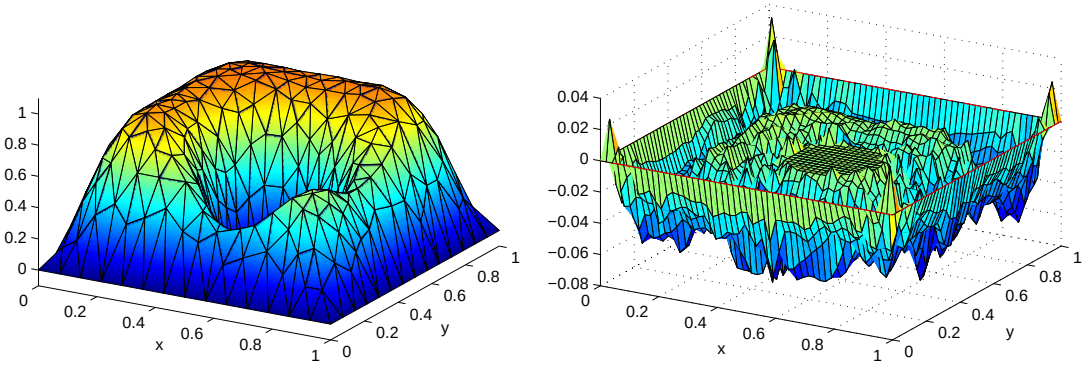
Rešitvi z metodama končnih razlik in končnih elementov

Na sliki 4.4 sta rešitev in napaka za primer z luknjo, dobljeni z MKR na enako gosti mreži kot v primeru brez luknje. MKR, ki na enostavni domeni deluje zelo dobro, ima v okolici luknje hude težave, saj je s pravilno mrežo točk ne more dobro predstaviti. Slika 4.5 prikazuje rešitev in napako, dobljeni z MKE na mreži s 662 elementi in 371 vozlišči. MKE v nasprotju z MKR domeno z luknjo reši skoraj enako natančno kot domeno brez luknje.

Sklep: MKR zelo dobro deluje le na domeni pravilne oblike. MKE dobro deluje tudi na nepravilni domeni.



Slika 4.4: Rešitev testnega primera z luknjo z MKR na mreži $20 \times 20 = 400$ točk (levo) in napaka (desno).



Slika 4.5: Rešitev testnega primera brez luknje z MKE na mreži s 662 elementi in 371 vozlišči (levo) in napaka (desno).

4.3 Parametri mreže prostih metod

Kot smo videli v prejšnjem poglavju, obstajajo številne različice mreže prostih metod, pri vsaki pa je treba pravilno izbrati kopico parametrov. Delno primerjavo med različicami najdemo v [Liu03], ki se predvsem posveča reševanju problemov v statiki, ožjo primerjavo med različicami metode MLPG pa v [AS02]. V tem delu se osredotočimo na različico MLPG1, ki je tipična predstavnica metod MLPG in je tudi dovolj splošna. Metodo smo najprej prototipno implementirali v programskem okolju Matlab z namenom preučiti vpliv različnih parametrov na natančnost metode in primerjati natančnost z MKR in MKE.

Metoda MLPG1 za poskusno funkcijo uporablja aproksimacijo MLS, ki zahteva izbiro naslednjih parametrov [Liu03]:

1. postopka generiranja vozlišč,
2. stopnje monomov, uporabljenih v aproksimaciji,
3. utežne funkcije za aproksimacijo,
4. postopka določanja velikosti nosilne domene dane točke glede na lokalno gostoto vozlišč,
5. brezdimenzijske velikosti α_S nosilne domene,
6. načina vsiljevanja zveznosti poskusne funkcije.

Poleg tega je za reševanje PDE z MLPG1 potrebno izbrati [AS02]:

7. testne funkcije W,
8. brezdimenzijsko velikost α_Q kvadraturne domene, s pomočjo katere določimo velikost kvadraturne domene na enak način kot velikost nosilne domene,
9. način vsiljevanja robnih pogojev,
10. dolžino časovnega koraka Δt,
11. postopek numerične integracije,
12. postopek reševanja dobljenega sistema linearnih enačb.

Nekateri izmed naštetih parametrov bodisi ne vplivajo bistveno na natančnost ali pa je za njih znana pravilna izbira, zato v teh primerih nismo podrobneje raziskovali različnih možnosti. Ostale smo eksperimentalno optimizirali za kar najmanjšo napako pri reševanju testnega primera brez luknje in izbiro preverili še na primeru z luknjo.

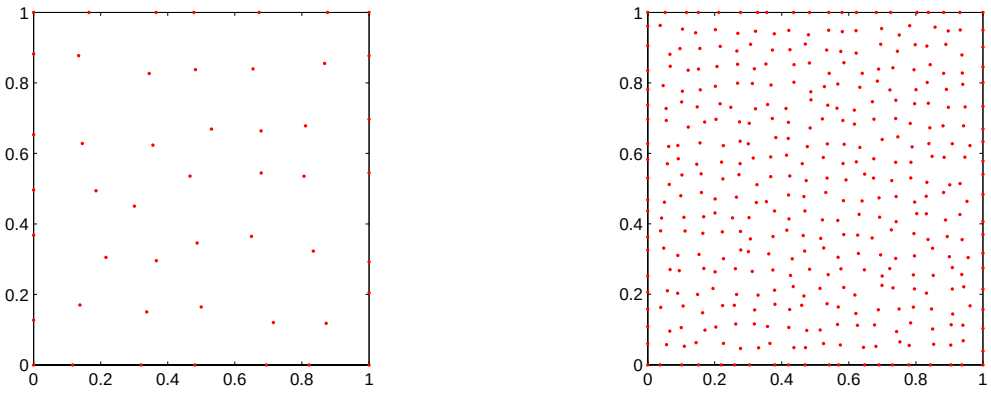
4.3.1 Parametri aproksimacije MLS

1. Generiranje vozlišč

Vozlišča smo generirali po naslednjem naključnem postopku:

1. Domeni Ω očrtaj kvadrat.
2. Na kvadrat postavi N vozlišč v pravilno mrežo $\sqrt{N} \times \sqrt{N}$. Razdalja med tako dobljenimi vozlišči je $d = a/(\sqrt{N} - 1)$, kjer je a stranica očrtanega kvadrata.
3. Odstrani vozlišča, ki padejo izven domene Ω ali so od njenega roba oddaljene manj kot 0.7 d.
4. Dodaj vozlišča na vse vogale domene.
5. Dodaj vozlišča na vse robove domene tako, da bo njihova medsebojna oddaljenost približno d.
6. Naključno premakni vsako notranje vozlišče za največ ± 0.3 d tako v smeri x kot y, vendar ne dopusti, da bi se vozlišče približalo robu na manj kot 0.7 d.
7. Naključno premakni vsako nevogalno robno vozlišče za največ ± 0.3 d v smeri, vzporedni z robom, na katerem leži.

Vsak poskus smo ponovili z dvema različnima semenoma naključnega generatorja in izbrali manj ugoden primer, to je tisti, pri katerem je bila neskončna norma napake večja. Postopek in seme smo torej izbrali tako, da bi dokazali sorazmerno neobčutljivost MPM na postavitev vozlišč. Slika 4.6 prikazuje diskretizacijo domene brez luknje z 49 in s 400 vozlišči.



Slika 4.6: Enotski kvadrat, diskretiziran z 49 vozlišči (levo) in s 400 vozlišči (desno).

2-3. Stopnja monomov in utežna funkcija

Bazne funkcije MLS smo konstruirali z monomi stopenj do 1 (metodo MLPG1 z uporabo tovrstnih baznih funkcij bomo krajše imenovali MPM1) in z monomi stopenj do 2 (MPM2). Za aproksimacijo smo uporabili utežno funkcijo iz enačbe 3.4 na strani 45.

4-5. Določanje velikosti nosilne domene in α_S

Preizkusili smo vse vrednosti parametra α_S od 1.8 do 3.2 s korakom 0.2. Parameter α_S predpisuje brezdimenzijsko velikost nosilne domene, s čimer implicitno predpiše neko *idealno število nosilnih vozlišč* n_I. MPM bo delovala dobro, če bodo imele vse točke, v katerih želimo ovrednotiti bazne funkcije, približno n_I nosilnih vozlišč. Razvili in preizkusili smo več postopkov, ki za dano točko $\mathbf{x}$ izračunajo velikost nosilne domene $d_S(\mathbf{x})$. V našem primeru ima nosilna domena obliko kroga, zato je d_S njen premer.

Postopek NumIdCorr (idealno število nosilnih vozlišč s popravkom). Najprej definirajmo: *koristna vozlišča* so vsa notranja vozlišča. Poleg tega sta za vsako daljico, ki predstavlja del roba, koristni dve vozlišči na tej daljici. Tretje in nadaljnja kolinearna robna vozlišča so nekoristna, saj kolinearnost povzroča težave pri izračunu koeficientov aproksimacije MLS. Postopek NumIdCorr poišče množico s toliko vozlišči, najbližjimi točki $\mathbf{x}$, da je med njimi n_I koristnih – razdaljo od $\mathbf{x}$ do zadnjega izmed njih označimo z $d(\mathbf{x}, n_I)$. Nato poišče najbližje vozlišče, ki je od $\mathbf{x}$ oddaljeno vsaj za $k \cdot d(\mathbf{x}, n_I)$ – razdaljo do tega označimo z $d(\mathbf{x}, n_{next})$. Polmer nosilne domene naj bo povprečje obeh razdalj, torej je njen premer:

$$d_S = 2r_S = d(\mathbf{x}, n_I) + d(\mathbf{x}, n_{next}). \tag{4.1}$$

Namen parametra k je zagotoviti, da je n_I-to koristno vozlišče nekoliko oddaljeno od roba nosilne domene, saj bi imelo tik ob robu utež 0. V naših poskusih smo uporabili vrednost $k = 1.01$.

Postopek AvgId (povprečna razdalja med idealnim številom vozlišč). Poišči $n_I + 1$ vozlišč, ki so najbližja točki $\mathbf{x}$, ne glede na njihovo koristnost. Povprečje razdalj do n_I-tega in $n_I + 1$-tega vozlišča vzemi za polmer vzorčne domene, torej je njen premer:

$$D_S = 2R_S = d(\mathbf{x}, n_I) + d(\mathbf{x}, n_I + 1).$$

Vzorčna domena bo vsebovala ravno n_I vozlišč. Nadaljuj po postopku, opisanem na strani 49: iz D_S izračunaj površino vzorčne domene $A_S = \frac{\pi}{4} D_S^2$. Nato po (3.14) izračunaj lokalno povprečno razdaljo med vozlišči $\overline{d}(\mathbf{x})$ in končno po (3.13) premer nosilne domene.

V primeru neokroglih nosilnih domen bi morali pri postopku NumIdCorr spremeniti mero razdalje pri iskanju najbližjih vozlišč. Postopek AvgId je neposredno uporaben za vse oblike domen, saj ni potrebno, da je vzorčna domena enake oblike kot nosilna.

Idealno število nosilnih vozlišč

Idealno število nosilnih vozlišč n_I izrazimo s parametrom α_S tako, da v enačbo 3.13 na strani 49 vstavimo $\overline{d}_{2D}$ iz enačbe 3.14. Dobimo velikost d_S nosilne domene, izraženo z

velikostjo D_S vzorčne domene in številom vozlišč n_{D_S} v slednji:

$$d_S = \alpha_s \frac{\sqrt{\frac{\pi}{4} D_S^2}}{\sqrt{n_{D_S}} - 1}.$$

Predpostavimo idealen primer – enakost nosilne in vzorčne domene $d_S = D_S$. Potem velja:

$$1 = \alpha_s \frac{\sqrt{\frac{\pi}{4}}}{\sqrt{n_{D_S}} - 1}. \tag{4.2}$$

Zaradi enakosti obeh domen je enaka tudi povprečna razdalja med vozlišči v obeh. Vzorčna domena torej predstavlja idealno oceno gostote vozlišč, to pa zagotavlja idealno število nosilnih vozlišč n_I. Zaradi enakosti domen obe vsebujeta tudi enako število vozlišč: $n_I = n_{D_S}$. Zdaj lahko izrazimo n_I iz (4.2):

$$\begin{aligned} \alpha_S &= \frac{\sqrt{n_I} - 1}{\frac{\sqrt{\pi}}{2}}, \\ n_I &= \left(\frac{\sqrt{\pi}}{2} \alpha_S + 1 \right)^2 = \frac{\pi}{4} \alpha_S^2 + \sqrt{\pi} \alpha_S + 1. \end{aligned} \tag{4.3}$$

6. Vsiljevanje zveznosti poskusne funkcije

Reševanje difuzijske enačbe z metodo MLPG1 ne zahteva zveznosti poskusne funkcije. Če vseeno želimo zvezno rešitev, lahko zveznost vsilimo z uporabo interpolacije povprečne razdalje, kot smo opisali na strani 50. Pri postopku AvgId lahko vsiljevanje zveznosti uporabimo neposredno, postopek NumIdCorr pa je treba nekoliko prilagoditi: iz (4.1) dobljeno vrednost vzamemo za premer vzorčne domene in nadaljujemo kot pri postopku AvgId.

Poskusili smo razviti tudi postopek določanja nosilne domene, ki bi zveznost zagotavljal neposredno, brez uporabe interpolacije.

Postopek AvgReg (povprečna razdalja med vozlišči, kot bi bila ta razvrščena v pravilno mrežo). Predpostavi, da so vsa vozlišča razvrščena v pravilno mrežo. V tem primeru je povprečna razdalja $\overline{d}$ med vozlišči v notranjosti domene Ω kar enaka $d = 1/(\sqrt{N} - 1)$. V bližini robov in vogalov je povprečna razdalja večja, ker del nosilne domene pade izven domene Ω, vendar je tudi tu $\overline{d}$ moč izračunati analitično. Nosilno domeno izračunaj neposredno iz povprečne razdalje, brez uporabe vzorčne domene. Postopek je uporaben le v primeru približno enakomerne razporeditve vozlišč.

Postopek AvgWeighted (utežena povprečna razdalja med vozlišči v vzorčni domeni, dobljeni s postopkom AvgReg). Za vzorčno domeno D_S izberi rezultat postopka AvgReg.

Povprečno razdaljo med vozlišči v tej vzorčni domeni izračunaj po spremenjeni enačbi 3.14, tako da prispevki vseh n_{D_S} vozlišč niso enaki, marveč uteženi glede na oddaljenost od točke $\mathbf{x}$. S tem vozlišča zvezno vstopajo in izstopajo iz vzorčne domene, zato sta zvezni tudi povprečna razdalja $\overline{d}$ in velikost d_S nosilne domene. Žal so poskusi pokazali, da število nosilnih vozlišč niha celo za faktor 50, zato je postopek neuporaben.

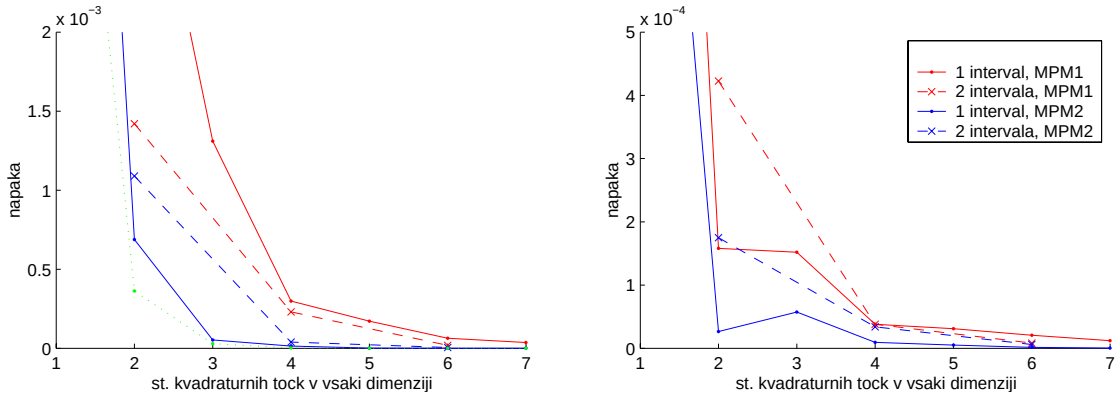
4.3.2 Parametri metode MLPG1

7-10. Testna funkcija, α_Q, vsiljevanje robnih pogojev in časovni korak

Testno funkcijo smo izbrali po enačbi 3.16 na strani 55. Preizkusili smo vse vrednosti parametra α_Q od 0.4 do 1.8 s korakom 0.2. Dirichletove robne pogoje smo vsiljevali s kolokacijo v robnih vozliščih. Večino poskusov smo izvedli s časovnim korakom $\Delta t = 0.125$, vpliv drugačnega koraka pa smo analizirali posebej.

11. Numerična integracija

Numerično integracijo potrebujemo za izračun vrednosti K^{int}, C^{int} in f^{int} v enačbi 3.22 na strani 61. Natančnost Gaussove kvadraturne formule, opisane na strani 17, smo preverili na primeru s 100 vozlišči. Levi del slike 4.7 prikazuje največjo absolutno napako v petih izbranih integralih v kvadraturni domeni izbranega vozlišča v notranjosti domene, to je $C^{int}_{43,41..45}$. Napake so narisane v odvisnosti od števila kvadraturnih točk v vsaki dimenziji. Točne vrednosti izbranih integralov C^{int} so med 10^{-5} in 10^{-2} in smo jih dobili z Matlabovo vgrajeno adaptivno funkcijo `dblquad` s toleranco 10^{-9}. Na desnem delu slike so največje absolutne napake integralov $K^{int}_{43,41..45}$. Točne vrednosti izbranih integralov K^{int} so med 10^{-5} in $3 \cdot 10^{-3}$.



Slika 4.7: Napaka v integraciji C^{int} in K^{int} v odvisnosti od števila kvadraturnih točk.

Polni črti na obeh grafih predstavljata običajno Gaussovo kvadraturno formulo [AS76], ki pri uporabljenih n točkah funkcijo interpolira s polinomom stopnje $2n-1$. Opazimo lahko,

da je integracija baznih funkcij MPM1 bistveno težja kot pri stopnji 2, kar je presenetljivo, saj so slednje bolj zapletene. Vzrok za ta pojav je, da so pri stopnji 1 nosilne domene majhne in s tem bazne funkcije stisnjene po oseh x in y, kvadraturne domene pa so velike. Funkcije zato integriramo prek večine njihovih prevojev in ekstremov. Pri stopnji 2 je optimalno razmerje med velikostjo nosilne in kvadraturne domene bistveno večje, zato funkcije integriramo le v bližini vrhov, kjer so razmeroma lepe. Zelena pikčasta črta na levem grafu predstavlja napako pri integraciji C^{int} z uporabo baznih funkcij MPM1, kjer pa smo uporabili takšno velikost nosilne in kvadraturne domene kot pri MPM2. V tem primeru je napaka integracije po predvidevanjih manjša kot pri stopnji 1. Žal bomo pozneje videli, da MPM1 s takima velikostima domen ni uporabna.
Sklep: Integracija baznih funkcij MPM1 je zahtevnejša kot pri MPM2.

Črtkani črti predstavljata integracijo, pri kateri integracijski interval razdelimo na dva intervala. Vsakega integriramo z uporabo $n/2$ točk z Gaussovo formulo stopnje $n-1$. Postopek posplošimo na dve in več dimenzij enako kot običajno Gaussovo formulo. Opazimo lahko, da se delitev izplača pri integraciji C^{int} z baznimi funkcijami MPM1, pri MPM2 pa ne. Prve so enostavnejše, zato integracija z mnogo točkami ni samo dobra – funkcijske vrednosti v kvadraturnih točkah namreč interpolira s polinomom visoke stopnje, ki zaradi številnih prenihajev slabše predstavlja originalno funkcijo kot enostavnejši polinom. Delitev na dva intervala in uporaba nižje stopnje v obeh se zato izplača. Bazne funkcije MPM2 pa so bolj zapletene, zato jih je bolje integrirati s polinomi višjega reda. Pri integraciji K^{int} se delitev na dva intervala ne izplača v nobenem primeru, saj v integralu nastopajo odvodi baznih funkcij, ki so po obliki bolj zapleteni.
Sklep: Bolje je uporabiti en interval z visokim redom Gaussove integracije kot več intervalov z nižjim redom.

Večino nadaljnjih poskusov smo izvedli z uporabo šestih kvadraturnih točk v vsaki dimenziji in brez delitve na dva intervala, vpliv drugačne integracije na končno rešitev pa smo analizirali posebej.

12. Reševanje linearnega sistema

Dobljeni nesimetrični linearni sistem smo najprej nepopolno razcepili [Hea02] na faktorja LU z Matlabovo funkcijo `luinc`, pri čemer smo zanemarili elemente, manjše od 10^{-6}. Približna faktorja smo uporabili za predpogojevanje pri reševanju sistema z metodo Bi-CGSTAB [Bic]. Od Matlabove funkcije `bicgstab` smo zahtevali takšno natančnost rešitve linearnega sistema, da bo relativni ostanek manjši od 10^{-6}. Pri tej natančnosti je bil prispevek netočnega reševanja sistema vedno vsaj za velikostni razred manjši od skupne napake v končni rešitvi.

Linearni sistem bi lahko reševali tudi z drugimi metodami. Izbira metode ne vpliva na natančnost končne rešitve, pač pa je pomembna s stališča časovne zahtevnosti in možnosti paralelizacije. V tem delu se ne ukvarjamo podrobneje z reševanjem sistema, saj gre

za zelo raziskano in v literaturi dobro opisano področje [Hea02, GO93, ŠT04, ŠT03]. Eno redkih možnosti za nadaljnje raziskave predstavlja prilagajanje metod multigrid za uporabo na sistemih, nastalih z MPM [Wes91, LOS04].

4.4 Rešitev testnih primerov z mreže prosto metodo

Slika 4.8 prikazuje rešitev testnega primera brez luknje (levo) in pripadajočo absolutno napako (desno). Uporabili smo 49 vozlišč in MPM1. Ostale parametre smo izbrali optimalno, kot bomo pokazali v razdelku 4.5. Opazimo lahko, da je napaka največja blizu robov, saj bazne funkcije MPM1 s tako malo vozlišči ne morejo predstaviti "divje" rešitve. Tako na robovih kot v notranjosti so v rešitvi tudi izbokline, ki so posledica oblike baznih funkcij. Uporaba MPM2 zmanjša napako, vendar še poudari izbokline (slika 4.9), povečanje števila vozlišč na 400 ob uporabi MPM1 pa omili vse slabosti (slika 4.10) in močno zmanjša napako. Z večanjem števila vozlišč se amplitude izboklin manjšajo, zato jih na sliki rešitve ni več moč opaziti, njihova frekvenca pa se veča, kar lahko vidimo na sliki napake. Pri 400 vozliščih opazimo tudi, da se kljub gladkosti rešitve močno razlikujejo parametri sosednjih vozlišč, predstavljeni z navpičnimi rdečimi črtami.

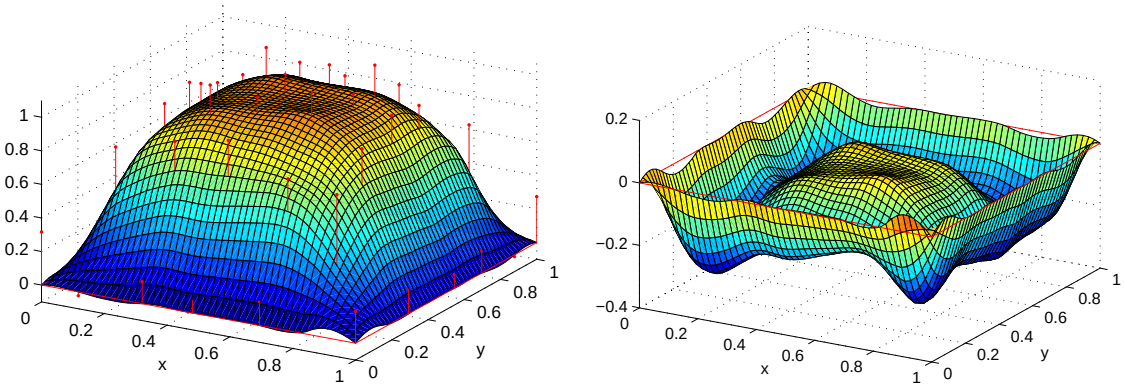
Rešitve in napake za testni primer z luknjo so prikazane na slikah 4.11-4.13. Tudi za ta primer veljajo iste ugotovitve kot za primer brez luknje, le absolutna vrednost napake je nekoliko večja.

4.5 Optimizacija parametrov mreže proste metode

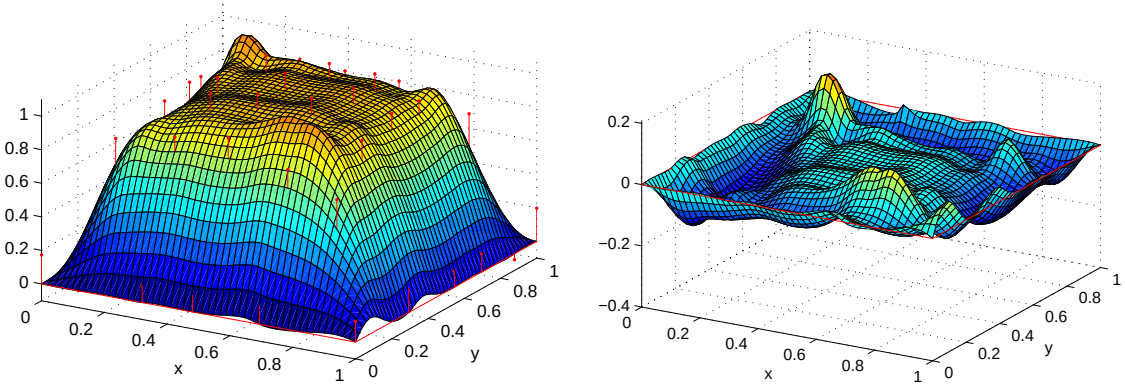
Parametre smo najprej optimizirali za kar najmanjšo napako na testnem primeru brez luknje. Kot kriterij smo vzeli neskončno normo absolutne napake rešitve, ovrednotene na mreži 102×102 točki. Celoten prostor parametrov je prevelik, da bi optimizirali vse hkrati, zato smo se morali odločiti za nek vrstni red. Najprej smo izbrali α_S, α_Q in postopek določanja povprečne razdalje med vozlišči, nato pa analizirali tudi vpliv časovnega koraka in numerične integracije. Med MPM1 in MPM2 ni vnaprej jasne zmagovalke, zato ves čas primerjamo obe. Na koncu razdelka v tabeli 4.5.3 povzamemo optimalne vrednosti vseh parametrov.

4.5.1 Parametra α_S in α_Q

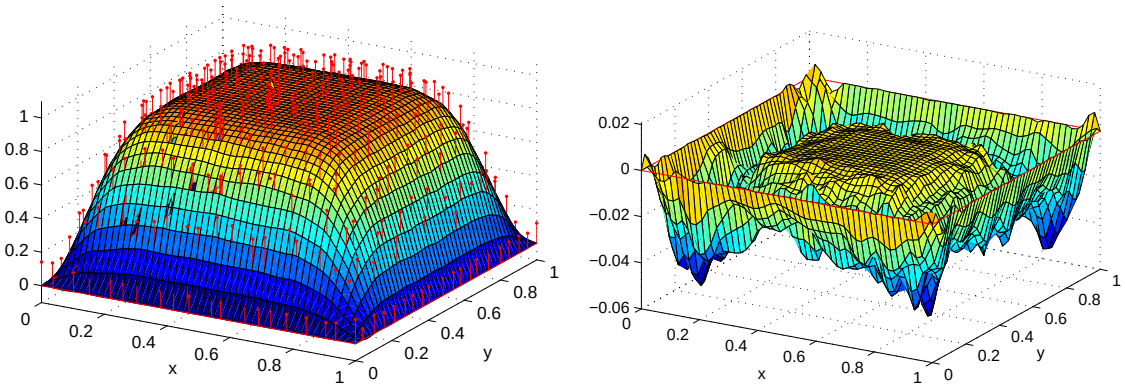
Parameter α_S mora zagotavljati, da ima vsaka točka primerno število nosilnih vozlišč, denimo dvakrat več od števila monomov, uporabljenih za aproksimacijo. Iz (4.3) lahko izračunamo primerno vrednost za α_S. Za MPM1 bi ta ocena pomenila $n_I = 6$ in $\alpha_S = 1.6$, vendar poskusi pokažejo, da je vrednost premajhna. Zaradi neenakomernega števila



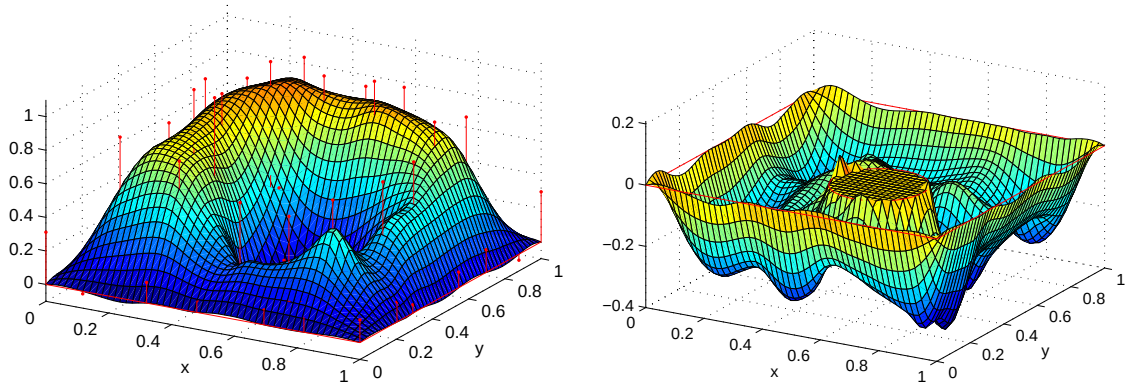
Slika 4.8: Rešitev testnega primera brez luknje z MPM1 z 49 vozlišči (levo); napaka rešitve (desno). Vozliščni parametri so na levi sliki prikazani kot rdeče palice s krogci na vrhu in so zaradi preglednosti narisani za 0.1 višje.



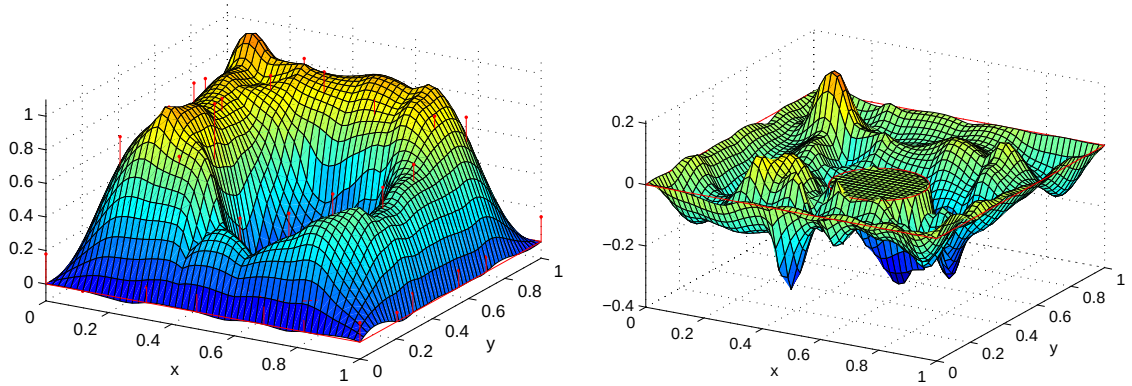
Slika 4.9: Rešitev testnega primera brez luknje z MPM2 z 49 vozlišči (levo); napaka rešitve (desno).



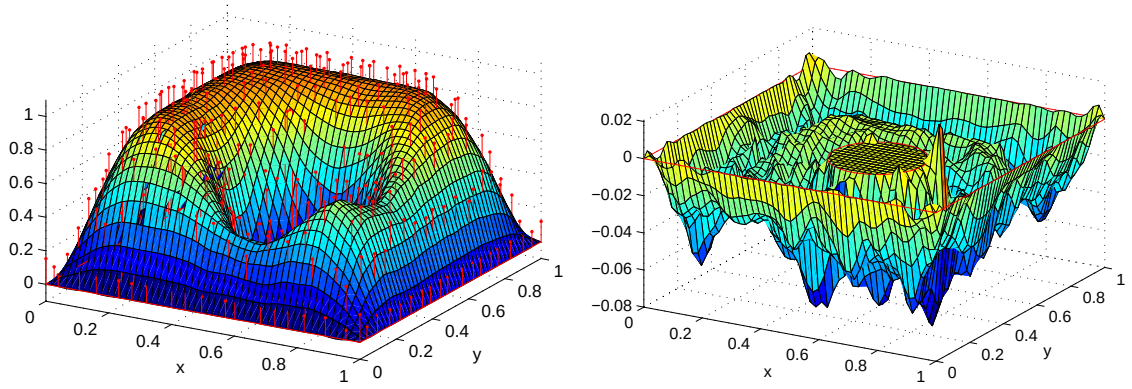
Slika 4.10: Rešitev testnega primera brez luknje z MPM1 s 400 vozlišči (levo); napaka rešitve (desno).



Slika 4.11: Rešitev testnega primera z luknjo z MPM1 s 53 vozlišči (levo); napaka rešitve (desno).



Slika 4.12: Rešitev testnega primera z luknjo z MPM2 s 53 vozlišči (levo); napaka rešitve (desno).

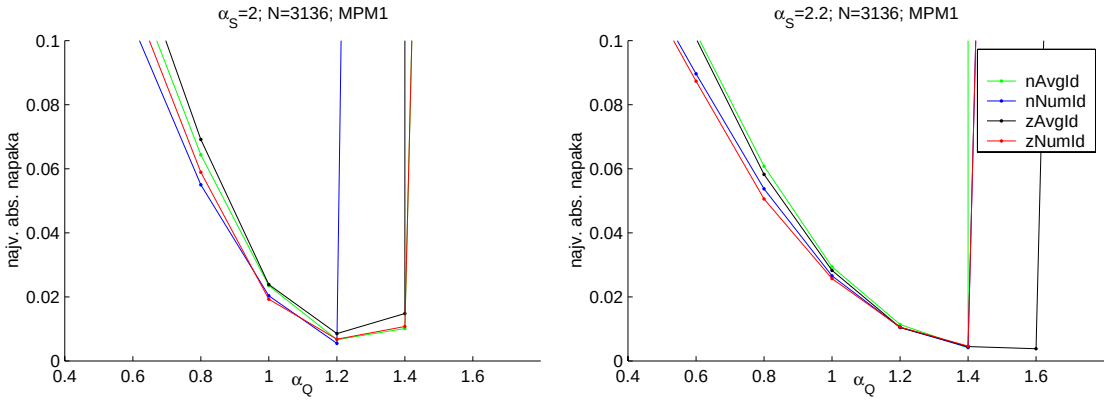


Slika 4.13: Rešitev testnega primera z luknjo z MPM1 z 356 vozlišči (levo); napaka rešitve (desno).

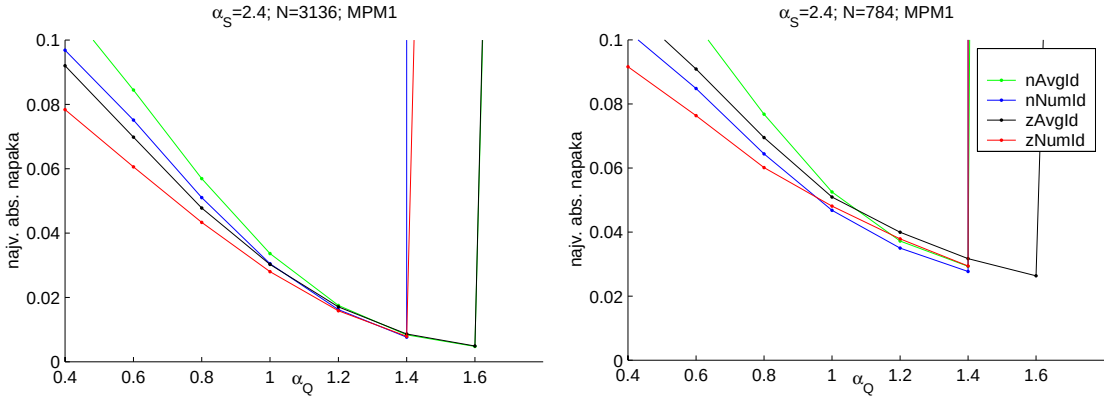
nosilnih vozlišč se pri tako majhnem α_S pogosto zgodi, da je njihovo število premajhno in je računanje koeficientov MLS slabo pogojeno. Za MPM2 s takšno grobo oceno dobimo $n_I = 12$ in $\alpha_S = 2.8$, kar je blizu optimalne vrednosti.

Pri obeh stopnjah smo najprej grobo preiskali širšo okolico teoretične vrednosti α_S in v [Liu03] priporočen interval $\alpha_Q = [0.4, 2]$. Opazili smo, da ima vsak parameter svoj *smiseln interval*, izven katerega so rešitve popolnoma neuporabne. V naslednji seriji poskusov smo podrobneje preiskali notranjost smiselnih intervalov.

Slike 4.14 levo, 4.14 desno in 4.15 levo prikazujejo odvisnost napake MPM1 od parametra α_Q za vrednosti α_S 2.0, 2.2 in 2.4, zaporedoma. Na tem mestu se še nismo želeli odločiti, ali bomo od baznih funkcij zahtevali zveznost in katero izmed dveh delujočih metod za določanje povprečne razdalje $\overline{d}$ med vozlišči bomo uporabili, zato smo poskuse naredili za vse štiri možnosti, ki jih bomo krajše imenovali *tip* baznih funkcij MLS. Vsak tip je na grafih predstavljen s svojo krivuljo. Prva črka v oznaki tipa je za nezvezne bazne funkcije



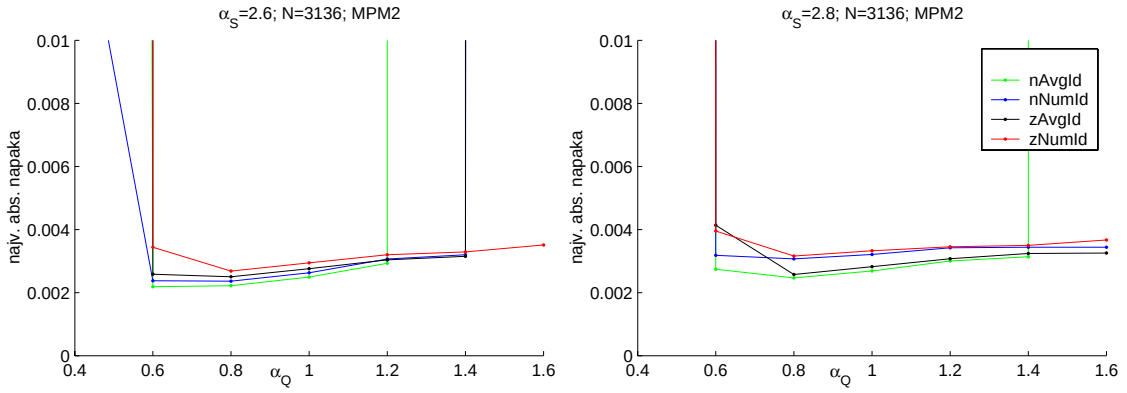
Slika 4.14: Odvisnost napake MPM1 od parametra α_Q s 3136 vozlišči ter vrednostima $\alpha_S = 2.0$ (levo) in $\alpha_S = 2.2$ (desno).



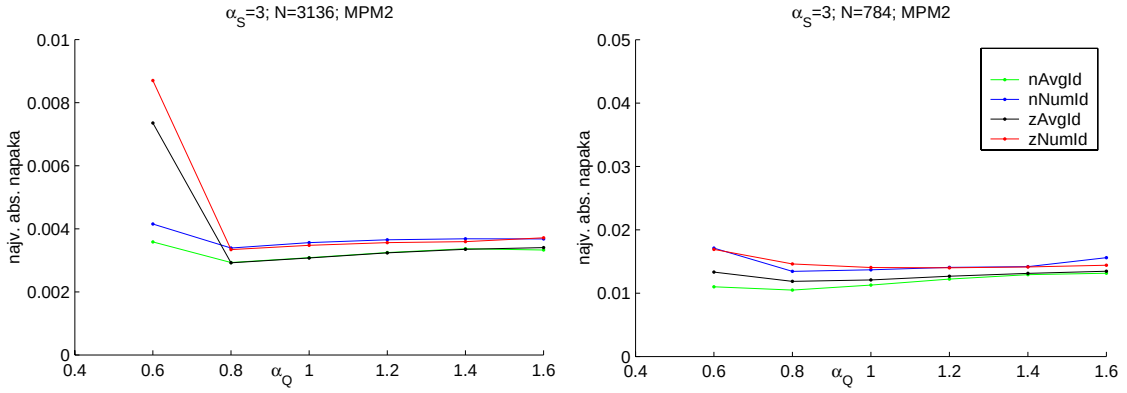
Slika 4.15: Odvisnost napake MPM1 od parametra α_Q z vrednostjo $\alpha_S = 2.4$ ter 3136 vozlišči (levo) in 784 vozlišči (desno).

enaka “n”, za zvezne pa “z”; preostanek je bodisi “AvgId”, če je $\overline{d}$ določena z metodo AvgId, in “NumId” pri metodi NumIdCorr. Desni del slike 4.15 se od levega razlikuje le v številu vozlišč, s čimer pokažemo, da razmere niso odvisne od števila vozlišč.

Najprej opazimo, da tip baznih funkcij MLS vpliva na širino smiselnega intervala α_Q. Ta je najširši pri tipu zAvgId, ki mu po vrsti sledijo nAvgId, zNumId in nNumId. Izmed preostalih dveh parametrov najprej izberemo $\alpha_S = 2.2$, kjer je napaka najmanjša. Pri vrednosti $\alpha_S = 2.0$ je napaka nekoliko večja, predvsem pa je bolj odvisna od tipa baznih funkcij in tudi smiselni intervali za α_Q so ožji. Pri vrednosti $\alpha_S = 2.4$ so smiselni intervali še nekoliko širši kot pri 2.2, vendar je napaka opazno večja. Na koncu določimo še $\alpha_Q = 1.4$, saj so pri tej vrednosti dobri vsi tipi baznih funkcij. Tip zAvgId deluje tudi z $\alpha_Q = 1.6$, vendar je pridobitev natančnosti zanemarljiva.



Slika 4.16: Odvisnost napake MPM2 od parametra α_Q s 3136 vozlišči ter vrednostima $\alpha_S = 2.6$ (levo) in $\alpha_S = 2.8$ (desno).

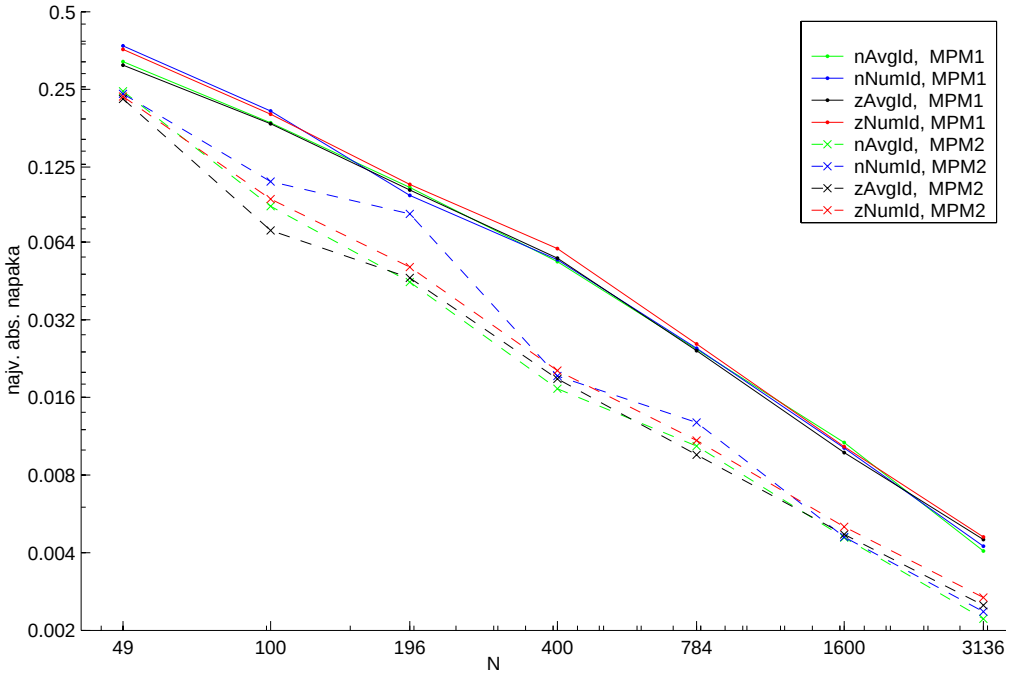


Slika 4.17: Odvisnost napake MPM2 od parametra α_Q z vrednostjo $\alpha_S = 3.0$ ter 3136 vozlišči (levo) in 784 vozlišči (desno).

Sliki 4.16 in 4.17 prikazujeta isto informacijo za MPM2, kjer so najboljše vrednosti za α_S okrog 2.8. Najprej opazimo, da je MPM2 bistveno manj občutljiva na izbiro parametrov od MPM1. Vrstni red tipov baznih funkcij glede na širino smiselnega intervala α_Q je

drugačen: zNumId, zAvgId, nNumId, nAvgId, vendar je spričo manjše odvisnosti napake od α_Q to manj pomembno. Napaka je najmanjša pri $\alpha_S = 2.6, \alpha_Q = 0.6$, vendar smo za nadaljnje poskuse raje uporabili $\alpha_S = 2.6, \alpha_Q = 0.8$, ki je skoraj enako dober, toda dlje od roba smiselnega intervala. Tipi baznih funkcij se pri MPM2 nekoliko bolj razlikujejo po natančnosti, najboljši je tip nAvgId.

Na sliki 4.18 nahajamo odvisnost napake od števila vozlišč za obe stopnji in vse štiri tipe baznih funkcij. Vrednosti α_S in α_Q sta izbrani optimalno za vsako stopnjo. Rešitev z večanjem števila vozlišč zelo lepo konvergira k točni rešitvi, saj je napaka približno obratno sorazmerna N. Razmerje med obema stopnjama je v grobem neodvisno od števila vozlišč. Pri MPM2 tip zAvgId daje najbolj konsistentno dobre rezultate.
Sklep: Napaka MPM je obratno sorazmerna številu vozlišč, oziroma sorazmerna kvadratu povprečne razdalje med vozlišči, torej je metoda drugega reda v prostoru. MPM1 ima pri enakem številu vozlišč približno dvakrat večjo napako kot MPM2. Priporočamo uporabo tipa baznih funkcij zAvgId. Optimalna parametra za MPM1 sta $\alpha_S = 2.2, \alpha_Q = 1.4$, za MPM2 pa $\alpha_S = 2.6, \alpha_Q = 0.8$, neodvisno od števila vozlišč.

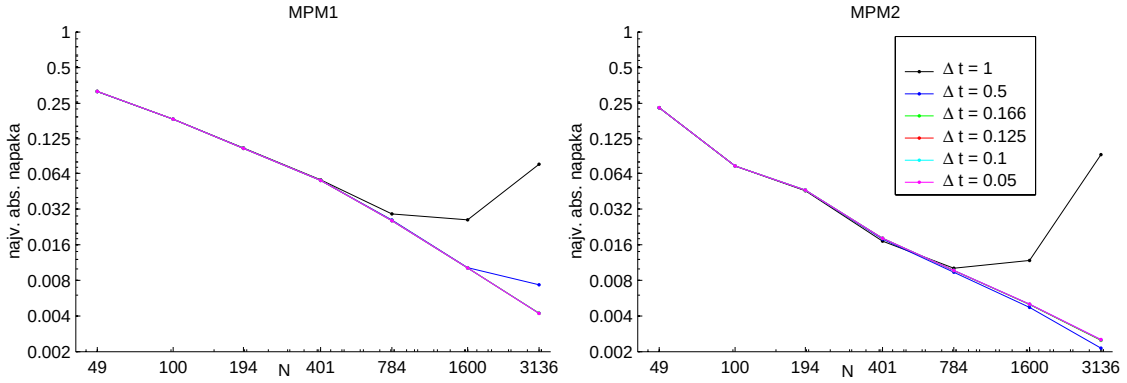


Slika 4.18: Napaka v odvisnosti od števila vozlišč za MLPG1 z baznimi funkcijami MLS obeh stopenj in vseh štirih tipov.

4.5.2 Časovni korak

Časovni korak ne sme biti prevelik, če naj ne vnaša dodatne napake. Čim manjša je razdalja med vozlišči, tem krajši korak si smemo privoščiti [Hea02]. Po drugi plati je od

dolžine koraka neposredno odvisna časovna zahtevnost, zato ga želimo čimbolj povečati. Z uporabljenim postopkom za generiranje vozlišč smo zagotovili, da se vozlišča na nobenem delu domene Ω ne morejo preveč zgostiti, saj bi to omejilo korak kljub manjši gostoti vozlišč v preostanku domene. MPM obeh stopenj smo preizkusili z različnimi časovnimi koraki, kot prikazuje slika 4.19. Med koraki, manjšimi ali enakimi 0.166, ni opaznih razlik v natančnosti, torej je za vse do tu predstavljene poskuse uporabljeni $\Delta t = 0.125$ primeren.



Slika 4.19: Napaka v odvisnosti od števila vozlišč in časovnega koraka za MPM1 (levo) in MPM2 (desno).

Pri MPM1 s korakom 1 natančnost trpi od 784 vozlišč naprej. Pri polovičnem koraku 0.5 se težave začnejo pri dvojni gostoti vozlišč, to je od 3136 vozlišč naprej. V prvem primeru je povprečna razdalja med vozlišči v celotni domeni $\overline{d} = \sqrt{1/784} = 1/28$, v drugem pa $1/56$.

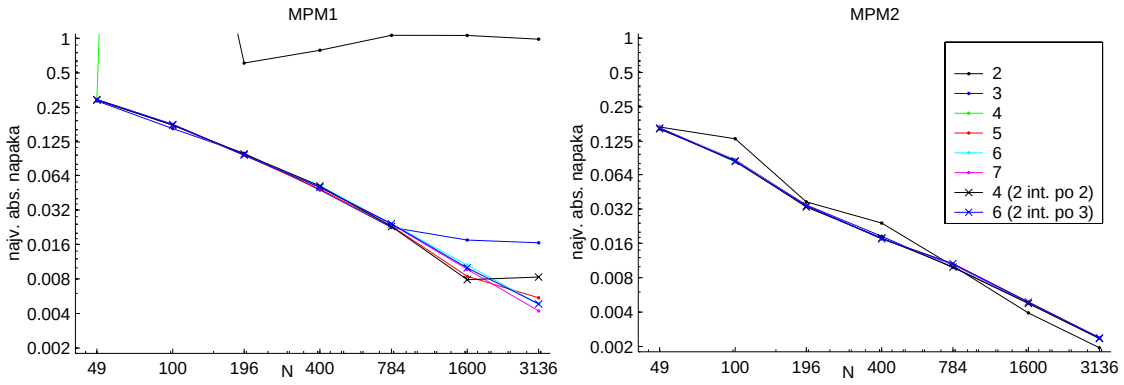
Sklep: Zaključimo lahko, da je za ta testni primer *dovoljeni časovni korak* oziroma varna zgornja meja časovnega koraka okrog $20\,\overline{d}$ za MPM1 in nekoliko več za MPM2. Dovoljeni časovni korak je premo sorazmeren z $\overline{d}$, ker je metoda tako v prostoru kot v času drugega reda natančnosti.

Pri MPM2 korak 1 odpove nekoliko pozneje kot pri MPM1. Presenetljivo je, da korak 0.5 z največjim številom vozlišč daje celo 20% nižjo napako kot vsi krajši koraki, prikazani na sliki, kar ni slučajno – napaka je najmanjša ravno pri koraku 0.5, potem pa s krajšanjem koraka rahlo raste. Enak pojav lahko zaznamo tudi pri manjšem številu vozlišč, vzrok zanj pa je, da vsak posamezni korak vnese neko dodatno napako, ki je posledica pretvorbe PDE v linearni sistem in reševanja sistema. Večje število narejenih korakov pomeni večjo končno napako. Da bi podprli to hipotezo, smo naredili dodatni poskus, v katerem smo močno povečali natančnost reševanja linearnega sistema. Tudi v tem primeru se je najbolje obnesel korak 0.5, vendar so se razlike rahlo zmanjšale. Sklepamo, da večino *napake koraka* predstavlja pretvorba PDE v linearni sistem.

Sklep: MPM z naraščanjem števila vozlišč kljub napaki koraka lepo konvergira k pravi rešitvi, torej je pri večjem številu vozlišč napaka koraka manjša.

4.5.3 Numerična integracija

Ugotoviti je treba najmanjše število kvadraturnih točk v vsaki dimenziji in s tem stopnjo Gaussove kvadraturne formule, ki še daje zadovoljive rezultate. Število kvadraturnih točk namreč določa časovno zahtevnost MPM. Slika 4.20 prikazuje napako v odvisnosti od števila vozlišč in števila kvadraturnih točk za MPM1 (levo) in MPM2 (desno).



Slika 4.20: Napaka v odvisnosti od števila vozlišč in števila kvadraturnih točk v vsaki dimenziji za MPM1 (levo) in MPM2 (desno).

Pri MPM1 je integracija z dvema točkama neuporabna, integracija s tremi in dvakrat po dvema točkama pa odpove pri večjem številu vozlišč. Uporabiti smemo 5, 6, 7 ali dvakrat po 3 točke. Integracija s 4 točkami je neuporabna ne glede na število vozlišč, česar nismo uspeli pojasniti. Ugotovili smo le, da:

- so uporabljene abscise in uteži kvadraturne formule pravilne,
- so rešitve neuporabne ne glede na seme naključnega generatorja vozlišč,
- rešitve rastejo v neskončnost, in sicer zlasti vozliščni parameter enega vozlišča,
- bazne funkcije v bližini tega vozlišča nimajo očitnih anomalij,
- je napaka v integraciji K^{int} in C^{int} za to vozlišče s 4 kvadraturnimi točkami le malenkost večja kot s 5 točkami,
- matrika dušenja in togostna matrika ne vsebujeta nobenih ekstremnih vrednosti,
- se lahko pojavu ognemo s pravilnejšo razporeditvijo vozlišč (vozlišča v postopku generiranja manj premikamo).

Za MPM2 smo že ugotovili, da je numerična integracija enostavnejša. Uporabne so vse prikazane metode, pri čemer pri velikem številu vozlišč dobimo nekoliko boljše rezultate z le dvema točkama v vsaki dimenziji. Kljub temu integracije z dvema točkama ne priporočamo, ker njeno obnašanje ni konsistentno.
Sklep: Pri MPM1 smemo uporabiti integracijo s 5, 6, 7 ali dvakrat po 3 točkami. Pri MPM2 smemo uporabiti 3 ali več točk.

Tabela 4.2: Optimalne vrednosti parametrov MPM za reševanje difuzijske enačbe.

parameter	optimalna vrednost za: MPM1	MPM2
α_S	2.2	2.6
postopek določanja $\overline{d}$	AvgId	
vsiljevanje zveznosti	da, z interpolacijo $\overline{d}$	
α_Q	1.4	0.8
št. točk v vsaki dimenziji za num. integr.	6	3

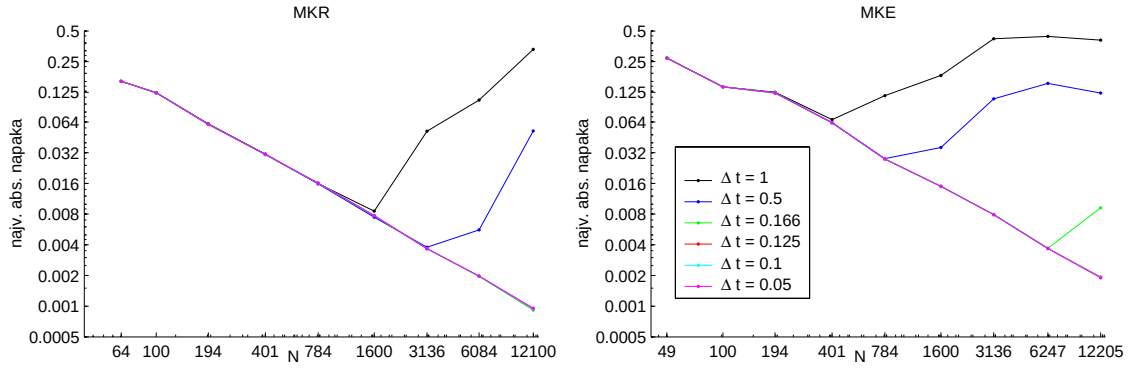
4.6 Primerjava z metodama končnih razlik in končnih elementov

V nasprotju z MPM imata MKR in MKE, kakršni uporabljamo v tem delu, le dva parametra: časovni korak Δt in število vozlišč.

4.6.1 Časovni korak

Slika 4.21 prikazuje odvisnost napake od števila vozlišč in Δt za MKR (levo) in MKE (desno). Podobno kot pri MPM se z večanjem števila vozlišč dovoljeni korak skrajšuje. Dovoljeni korak se pri treh metodah različno izraža z $\overline{d}$, ker ga ne omejuje povprečna razdalja med vozlišči, marveč najmanjša. Ta je pri MKR kar enaka povprečni, pri MKE pa veliko manjša.
Sklep: Pri MKR je dovoljeni korak približno $35\,\overline{d}$, pri MKE pa približno $15\,\overline{d}$.

Podobno kot pri MPM2 je tudi pri MKR prisoten vpliv napake koraka, zaradi katere se pri zelo majhnih korakih napaka v končni rešitvi PDE poveča. Pri največjem prikazanem številu vozlišč je napaka s korakom 0.166 približno za 5 % manjša kot s korakom 0.05. Pri MKE in MPM1, ki sta obe le prvega reda konsistentnosti, tega pojava ni.



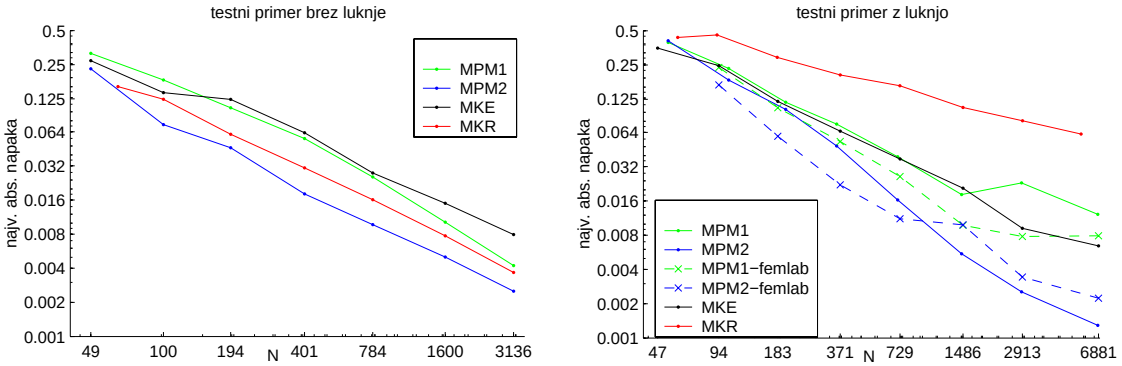
Slika 4.21: Napaka v odvisnosti od števila vozlišč in Δt za MKR (levo) in MKE (desno).

4.6.2 Primerjava natančnosti metod na testnih primerih

Levi graf na sliki 4.22 prikazuje primerjavo med vsemi tremi metodami na testnem primeru brez luknje. Pri MPM so bili uporabljeni optimalni parametri, Δt pa je bil v vseh primerih 0.125.
Sklep: MKR se na tako preprostem primeru obnese zelo dobro. MKE in MPM1 sta približno enakovredni, MPM2 pa je precej boljša od vseh ostalih. Pri vseh metodah je napaka približno obratno sorazmerna številu vozlišč.

Na testnem primeru z luknjo (slika 4.22 desno) MKR v skladu s pričakovanji omaga. MKE je podobno dobra kot na primeru brez luknje. Paket Femlab na primeru z luknjo generira mrežo, ki ima v bližini luknje približno za 40 % zgoščena vozlišča. Pri obeh MPM smo najprej preizkusili delovanje na z opisanim naključnim postopkom dobljenih vozliščih, ki smo jih v bližini luknje prav tako zgostili (polni črti na grafu), nato pa še z vozlišči, uporabljenimi za MKE (črtkani črti). Rezultati se precej razlikujejo, zato težko priporočimo eno ali drugo možnost.

Z MPM2 in 1486 vozlišči dobimo presenetljivo veliko napako, celo večjo kot z MPM1, kar je posledica neugodne razporeditve vozlišč. Če eno izmed vozlišč na delu domene, kjer je napaka velika, premaknemo za $0.1\overline{d}$ v poljubno smer, se napaka MPM2 skoraj prepolovi, medtem ko napaka MPM1 ostane podobna.
Sklep: Tudi na primeru z luknjo je MPM1 primerljiva z MKE, vendar njena napaka bolj niha. MPM2 je bistveno natančnejša od obeh in se zdi nekoliko manj občutljiva na izbiro vozlišč od MPM1.



Slika 4.22: Napaka v odvisnosti od števila vozlišč za vse tri metode na testni domeni brez luknje (levo) in z luknjo (desno).

Primerjava glede na število vozlišč ni poštena, saj se metode bistveno razlikujejo po zapletenosti formulacije, vložku človeškega dela pri reševanju danega problema in po računski zahtevnosti. V naslednjem poglavju bomo primerjali časovno zahtevnost metod pri zahtevani natančnosti, kar bolj zanima končnega uporabnika. Obravnavati moramo predvsem primer z luknjo, ki je blizu realnim simulacijam, vendar je dogajanje preveč

zapleteno, da bi lahko iz podatkov na sliki 4.22 dobili uporabno eksplicitno formulo za oceno napake v odvisnosti od števila vozlišč.

Sklep: V grobem lahko zaključimo, da za isto natančnost MKR potrebuje približno desetkrat več vozlišč kot MKE, MPM1 jih potrebuje približno toliko kot MKE, MPM2 pa dva- do štirikrat manj kot MKE.

Poglavje 5

Izvedba in časovna zahtevnost

V tem poglavju najprej opišemo našo izvedbo metod končnih razlik, končnih elementov in mreže proste metode. Za vsako metodo podamo psevdokodo celotnega algoritma za gradnjo sistema navadnih enačb in analiziramo njegovo asimptotično časovno in prostorsko zahtevnost. Vsi uporabljeni algoritmi so asimptotično optimalni. Nato eksperimentalno ocenimo natančno časovno zahtevnost MPM in si ogledamo vpliv parametrov na čas izvajanja posameznih gradnikov. Eksperimentalno ocenimo tudi natančno časovno zahtevnost MKE in MKR ter ju primerjamo z MPM z upoštevanjem razlik v natančnosti metod. Dotaknemo se tudi človeškega dela, potrebnega pri reševanju PDE z vsako izmed metod. Na koncu poglavja analiziramo tiste lastnosti linearnih sistemov, dobljenih z vsemi tremi metodami, ki vplivajo na časovno zahtevnost njihovega reševanja, pri čemer se ne spuščamo v samo reševanje sistema.

5.1 Metoda končnih razlik

Linearni sistem za reševanje difuzijske enačbe z MKR sestavimo z enostavnim postopkom, katerega vhodni podatki so: opis domene Ω, prostorski in časovni korak Δx in Δt ter difuzijska konstanta c.

ALGORITEM SESTAVISISTEMMKR($\Omega, \Delta x, \Delta t, c$)

1 $V \leftarrow$ vozlišča z ekvidistantne mreže, oddaljena med seboj za Δx, ki pokrijejo Ω in njene robove Γ
2 $B \leftarrow$ robna vozlišča (vozlišča iz V, ki so bodisi izven Ω ali v Ω, vendar oddaljena od roba manj kot $\Delta x/2$)
3 $seznamA, seznamB$//seznama, v katera dodajamo indekse in vrednosti neničelnih elementov matrik A in B
4 **for each** $\mathbf{x}_i \in V$
5 **do if** $\mathbf{x}_i \in B$
6 **then** $seznamA$.DODAJ$(i, i, 1)$

```
 7              seznamB.DODAJ(i, i, 1)
 8        else  seznamA.DODAJ(i, i, 2 + 4cΔt/(Δx)^2)
 9              seznamB.DODAJ(i, i, 2 − 4cΔt/(Δx)^2)
10              S ← indeksi 4 sosedov vozlišča x_i
11              for each j in S
12              do seznamA.DODAJ(i, j, −cΔt/(Δx)^2)
13                 seznamB.DODAJ(i, j, cΔt/(Δx)^2)
14  A ← SESTAVIREDKOMATRIKO(seznamA, |V|)
15  B ← SESTAVIREDKOMATRIKO(seznamB, |V|)
```

Funkcija SESTAVIREDKOMATRIKO($seznam, N$) sestavi matriko velikosti $N \times N$, katere neničelni elementi so v seznamu podani kot trojke $< vrstica, stolpec, vrednost >$. Če seznam vsebuje več elementov z isto številko vrstice in stolpca, kar se sicer pri MKR ne more zgoditi, se pripadajoče vrednosti seštejejo. Vrednosti posameznih elementov sledijo iz enačbe 2.14 na strani 26.

Množic V in B ni potrebno dejansko generirati. Število $|V|$ vozlišč bomo kot doslej označevali z N, število $|B|$ robnih vozlišč pa je sorazmerno $\sqrt{N}$. Predpostavimo, da nam opis domene Ω omogoča preverjanje pogoja v vrstici 5 v konstantnem času. Zahtevnost zanke v vrsticah 4-13 je potemtakem $O(5(|V| - |B|) + |B|) = O(N)$.

5.1.1 Časovna zahtevnost Matlabove funkcije `sparse`

Za sestavljanje redke matrike smo uporabili Matlabovo funkcijo `sparse` [Matb]. Princip njenega delovanja ni javno objavljen, vemo pa, da Matlab redke matrike shranjuje kot *stik stolpcev*, stolpce pa kot tabelo parov $< indeks, vrednost >$, urejeno po indeksih [GMS92]. Elementi (i, j) so lahko v vhodnem seznamu podani v poljubnem zaporedju, zato jih je treba predhodno urediti po položaju v matriki $p = Ni + j$, saj bi morali sicer vsakega posebej vrivati v obstoječo tabelo.

V našem primeru je število elementov v vseh treh metodah sorazmerno z N. Gradnja redke matrike iz urejenega seznama elementov traja le $O(N)$ časa, zato je ob predpostavki, da je uporabljen kateri izmed običajnih algoritmov za urejanje [CLRS01], na primer *urejanje s kopico* (angl. heap sort) ali *urejanje s porazdelitvami* (angl. quicksort), skupna asimptotična časovna zahtevnost funkcije `sparse` enaka $O(N \log N)$. Takšno zahtevnost so potrdile tudi meritve.

5.1.2 Skupna časovna in prostorska zahtevnost

Skupna časovna zahtevnost je enaka $T_{MKR} = O(N + N \log N + N) = O(N \log N)$. Prostorska zahtevnost vrstic 4-13 je približno enaka prostoru za shranjevanje obeh seznamov, to je $2 \cdot 5 \cdot 3N \cdot e$. Z e smo označili prostor za hranjenje enega števila v plavajoči vejici, kar v dvojni natančnosti pomeni 8 bajtov [Kod00]. Prostor za spremenljivke, katerih velikost

ni odvisna od N, lahko zanemarimo. Vsaka izmed matrik zaseda podobno velik prostor kot seznam, iz katerega je bila zgrajena [GMS92], pri čemer lahko seznam po gradnji posamezne matrike zavržemo. Skupna prostorska zahtevnost je torej enaka prostoru za shranjevanje dveh seznamov in ene matrike: $S_{MKR} \approx 45N \cdot e \approx 360N$ bajtov.

Tako pri MKR kot pri ostalih dveh metodah bi se dalo prihraniti nekaj prostora, vendar to ne bi bilo vredno truda, saj prostor, potreben za gradnjo sistema, pri nobeni od metod ni omejujoč faktor. Navedene prostorske zahtevnosti so torej le orientacijske in veljajo za našo izvedbo metod.

5.2 Metoda končnih elementov

Domeno Ω je treba najprej razdeliti na končne elemente oziroma *triangulirati*, česar v dveh dimenzijah ni težko avtomatizirati. MKE se pogosto uporablja z zmernim številom vozlišč, zato zadostuje algoritem za triangulacijo po Delaunayevem kriteriju s kvadratno časovno zahtevnostjo [BE95], kakršnega uporablja tudi Femlabova funkcija `meshinit` [Com]. Pri večjem številu vozlišč bi morali nujno uporabiti katerega izmed bolj zapletenih algoritmov [BE95] s časovno zahtevnostjo $O(N \log N)$.

Linearni sistem za MKE sestavimo lokalno. Iz enačbe 2.20 na strani 31 izrazimo lokalni matriki:

$$\begin{aligned} C^L_{k,l} &= \int_A \phi_k \phi_l dA, \\ K^L_{k,l} &= c \int_A \left[\phi_{k,x}\phi_{l,x} + \phi_{k,y}\phi_{l,y}\right] dA, \end{aligned}$$

kjer je A površina trikotnega elementa. Za izračun integrala C uporabimo analitično formulo iz [Buc95] za integrale lokalnih baznih funkcij FEM prvega reda s trikotnimi elementi:

$$\int_A \phi_1^\alpha \phi_2^\beta \phi_3^\gamma dA = \frac{\alpha!\beta!\gamma!}{(\alpha+\beta+\gamma)!} 2A \qquad \forall \alpha, \beta, \gamma \in \{0, 1, \ldots\}. \tag{5.1}$$

Torej velja:

$$C^L_{k,l} = \begin{cases} 2A & k = j \\ A & \text{sicer.} \end{cases}$$

Tudi K^L izračunamo analitično, saj so prostorski odvodi lokalnih baznih funkcij konstante.

Vhodni podatki postopka za sestavljanje sistema z MKE so: seznam vozlišč $V = \{< x, y, robni >\}$ (koordinati vozlišča in podatek, ali gre za robno vozlišče), seznam elementov $E = \{< i_1, i_2, i_3 >\}$ (indeksi treh vozlišč iz seznama V, ki sestavljajo element, našteti v pozitivni smeri), časovni korak Δt in difuzijska konstanta c. Domena Ω je implicitno opisana z elementi in vozlišči.

ALGORITEM SESTAVISISTEMMKE$(V, E, \Delta t, c)$

1 $seznamA, seznamB$//seznama, v katera dodajamo indekse in vrednosti neničelnih elementov matrik A in B
2 **for each** $e \in E$
3 **do for** $k \leftarrow 1$ **to** 3
4 **do if** vozlišče št. $e.i_k$ je robno
5 **then if** robno vozlišče št. $e.i_k$ še ni obdelano
6 **then** $seznamA$.DODAJ$(e.i_k, e.i_k, 1)$
7 $seznamB$.DODAJ$(e.i_k, e.i_k, 1)$
8 vozlišče je zdaj obdelano
9 **else for** $l \leftarrow 1$ **to** 3
10 **do** $seznamA$.DODAJ$(e.i_k, e.i_l, 2C^L_{k,l} + \Delta t K^L_{k,l})$
11 $seznamB$.DODAJ$(e.i_k, e.i_l, 2C^L_{k,l} - \Delta t K^L_{k,l})$
12 $A \leftarrow$ SESTAVIREDKOMATRIKO$(seznamA, |V|)$
13 $B \leftarrow$ SESTAVIREDKOMATRIKO$(seznamB, |V|)$

Časovna zahtevnost zanke v vrsticah 2-11 je $O(|E|)$, vsak prehod pa doda 9 elementov v vsakega izmed seznamov oziroma nekaj manj za robna vozlišča. Skupna časovna zahtevnost je zato $O(|E| + 9|E| \log 9|E|) = O(|E| \log |E|)$. Vsak element ima 3 vozlišča, vozlišča pa v povprečju nastopajo v s elementih, kjer je s tipično med 3 in 10 in ni odvisen od N [Kat03]. Dolžina obeh seznamov je torej $9|E| = 3sN$, skupna asimptotična časovna zahtevnost pa $T_{MKE} = O(N \log N)$.

5.2.1 Izvedba v jeziku C++

Izdelali smo tudi optimizirano izvedbo MKE v jeziku C++ [Str97], zato smo morali implementirati podatkovno strukturo za redke matrike. Odločili smo se za nekoliko manj splošno predstavitev od Matlabove, kajti naše matrike vsebujejo neničelne elemente v večini vrstic. Poleg tega je vnaprej znana velikost matrik in približno število neničelnih elementov, ki so tudi približno enakomerno razporejeni med vrstice. Redko matriko smo zato predstavili kot tabelo redkih vrstičnih vektorjev. Vsak izmed njih je objekt razreda `RedekVektor` in vsebuje dve tabeli spremenljive velikosti, ki hranita urejen seznam indeksov, to je številk stolpcev, kjer se nahajajo neničelni elementi, in pripadajoče vrednosti.

V obstoječ redek vektor dodamo nov element tako, da ga vrinemo na ustrezno mesto v urejeni tabeli. Časovna zahtevnost dodajanja je $O(s')$, kjer je s' število obstoječih neničelnih elementov v vrstici, v katero dodajamo nov element. Pri sestavljanju sistema z MKE je v povprečju $s' = 3s/2$. Predhodno urejanje elementov ni potrebno. Dodajanje vseh $3sN$ elementov v redko matriko potemtakem zahteva $O(3sN \cdot 3s/2)) = O(s^2N) = O(N)$, saj je s konstanta. Tako je tudi skupna zahtevnost $T^{C++}_{MKE} = O(N)$.

Namesto tabele bi lahko uporabili kazalčni seznam [CLRS01], v katerem bi ustrezno mesto poiskali z linearnim iskanjem v času $O(s'/2)$, element pa bi vrinili v konstantnem času $O(1)$. Za seznam se nismo odločili zaradi zahtevnejše režije in bolj zapletene komunikacije v vzporedni različici programa.

5.2.2 Prostorska zahtevnost

Za seznam V potrebujemo približno $3N \cdot e$, za E pa $3|E| \cdot e$ prostora. Vsak izmed seznamov za gradnjo matrik ima $3sN$, vsaka izmed matrik pa sN elementov. Seznama za gradnjo matrik lahko po uporabi zavržemo, seznamov V in E pa ne, ker jih bomo potrebovali tudi za ovrednotenje rešitve v točkah, kjer nas zanima njena vrednost. Skupna prostorska zahtevnost je torej:

$$\begin{aligned} S_{MKE} &\approx (3N + 3sN/3 + 2 \cdot 3 \cdot 3sN + 3sN)e \\ &\approx (3 + 22s)Ne \\ &\approx 500N \text{ do } 2000N \text{ bajtov.} \end{aligned}$$

5.3 Mreže proste metode

Vhodni podatki za gradnjo linearnega sistema z MPM so: seznam vozlišč $V = \{< x, y, indeks, robni >\}$ (koordinati vozlišča, njegov indeks v seznamu V in podatek, ali gre za robno vozlišče), časovni korak Δt, difuzijska konstanta c in parametri MPM, našteti v poglavju 4.3. Vozlišča lahko generiramo bodisi z istim algoritmom kot pri MKE ali z enostavnejšim, kakršnega smo opisali v prejšnjem poglavju. Podobno kot pri MKR matriki A in B sestavljamo po vrsti od prve do zadnje vrstice.

ALGORITEM SESTAVISISTEMMPM($V, \Delta t, c,$ parametri)

```
1  seznamA, seznamB//seznama, v katera dodajamo indekse in vrednosti neničelnih elementov matrik A in B
2  for each x_i ∈ V
3  do if vozlišče x_i je robno
4        then S ← množica vozlišč, ki ležijo v nosilni domeni Ω_Si vozlišča x_i
5             for each x_S ∈ S
6             do seznamA.DODAJ(x_i.indeks, x_S.indeks, φ_{x_S}(x_i))
7        else SQ ← množica vozlišč v nosilni domeni Ω_SQi kvadraturne domene x_i
8             for each x_SQ ∈ SQ
9             do seznamA.DODAJ(x_i.indeks, x_SQ.indeks, 2C^int_{x_i,x_SQ} + ΔtK^int_{x_i,x_SQ})
10                seznamB.DODAJ(x_i.indeks, x_SQ.indeks, 2C^int_{x_i,x_SQ} − ΔtK^int_{x_i,x_SQ})
11 A ← SESTAVIREDKOMATRIKO(seznamA, |V|)
12 B ← SESTAVIREDKOMATRIKO(seznamB, |V|)
```

Uporabili smo okrajšavo $\phi_{\mathbf{x}} = \phi_{\mathbf{x}.indeks}$ za bazno funkcijo, ki pripada vozlišču $\mathbf{x}$, to je, ima vrh v vozlišču $\mathbf{x}$. Podobno okrajšavo smo uporabili za elemente matrik C^{int} in K^{int} iz enačbe 3.22 na strani 61. V nasprotju s prejšnjima metodama pri MPM vrstice B, ki pripadajo robnim vozliščem, nimajo enice na diagonali – njihova enačba se namreč ne glasi $u_i^{(k+1)} = u_i^{(k)}$ kot pri MKR in MKE, marveč je navedena v enačbi 3.18 na strani 58, robni pogoj pa je shranjen v f_i.

Postopek smo navedli zelo poenostavljeno, kajti večina kompleksnosti se skriva v:

1. iskanju vozlišč v nosilni domeni (vrstici 4 in 7),
2. računanju vrednosti baznih funkcij (vrstica 6),
3. Gaussovi integraciji C^{int} in K^{int} (vrstici 9 in 10).

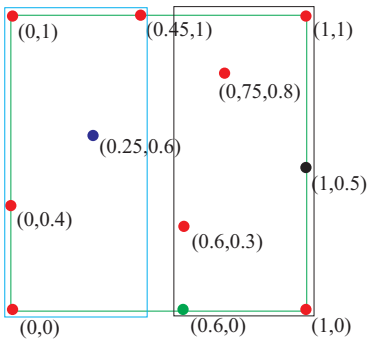
V nadaljevanju podrobneje opišemo vse tri sestavne dele in povzamemo celoten postopek, pri čemer se naslanjamo na našo izvedbo v jeziku C++.

5.3.1 Iskanje vozlišč v nosilni domeni

Vozlišča moramo hraniti v podatkovni strukturi, ki poleg iteracije čez vsa vozlišča omogoča tudi učinkovito iskanje vozlišč, ki ležijo znotraj danega kroga – nosilne domene. Odločili smo se za k-D drevo [FBF77, BKOS00], katerega drevesna vozlišča imajo naslednjo strukturo:

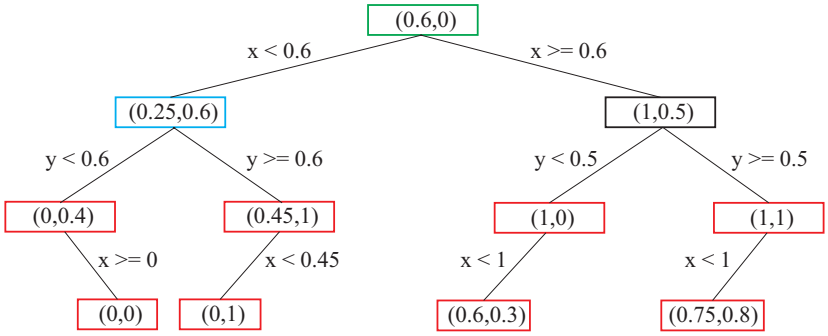
```
class KDDrevo::public Vozlisce { //deduje polja x, y, indeks in robni
    int globina; //0 pri korenu, 1 pri njegovih sinovih...
    Pravokotnik podrocje; //najmanjsi pravok., ocrtan vsem vozliscem v tem poddrevesu
    KDDrevo *levi, *desni; //poddrevesi (NULL pri listih)
};
```

Koren drevesa vedno predstavlja vozlišče $\mathbf{x}_R$, katerega x-koordinata je najmanjša izmed vseh x-koordinat, večjih od ali enakih *mediani*. Na primer med vozlišči $(0, y_1)$, $(0.4, y_2)$, $(0.6, y_3)$ in $(0.6, y_4)$ je mediana 0.5. Koren je lahko katerokoli izmed vozlišč z x-koordinato 0.6. Levo poddrevo vsebuje vsa vozlišča z x-koordinatami, manjšimi od x_R, desno pa vsa vozlišča s koordinatami, večjimi ali enakimi od x_R, razen vozlišča $\mathbf{x}_R$. Enako se vozlišča delijo na vseh sodih globinah, medtem ko se na lihih globinah delijo po y-koordinatah. Takšna definicija zagotavlja uravnoteženost drevesa [BKOS00], razen če imajo vsa vozlišča isto x-koordinato in v podobnih izrojenih primerih, do kakršnih v MPM ne sme priti. Slika 5.1 prikazuje primer 11 vozlišč v enotskem kvadratu, slika 5.2 pa iz njih zgrajeno k-D drevo. Drevesna vozlišča na globinah 0 in 1 so obarvana enako kot njihov očrtan pravokotnik na sliki 5.1.



Slika 5.1: 11 vozlišč, uporabljenih za k-D drevo na sliki 5.2.

Obstajajo tudi druge različice drevesa, na primer takšna, ki vozlišča shranjuje le v listih drevesa, ne pa tudi v notranjih drevesnih vozliščih. V tem primeru se lahko pri neugodnih



Slika 5.2: k-D drevo, zgrajeno iz vozlišč na sliki 5.1.

podatkih zgodi, da drevo po zgornji definiciji ne obstaja. Predpostavimo na primer, da je potrebno zgraditi (pod)drevo, ki naj vsebuje spodnja tri vozlišča v črnem pravokotniku na sliki 5.1 – (0.6,0), (0.6,0.3) in (1,0). Mediana po x je 0.6, zato gredo vsa tri vozlišča v desno poddrevo. Na naslednji globini je mediana po y enaka 0, zato gredo spet vsa tri vozlišča v desno poddrevo in postopek pade v neskončno rekurzijo. Pri tej različici drevesa bi bilo zategadelj potrebno spremeniti definicijo, katera vozlišča spadajo v katero poddrevo, tako da bi posebej obravnavali takšne neugodne primere [BKOS00].

Gradnja drevesa

Vhodni podatki *rekurzivnega* algoritma za gradnjo drevesa so: drevesno vozlišče, ki predstavlja *koren* drevesa, in seznama $vozlUrPoX$ in $vozlUrPoY$. V vsakem izmed seznamov so našteta vsa vozlišča, urejena po njihovih x- oziroma y-koordinatah. Urejenost omogoča enostavno iskanje mediane. Elemente seznama štejemo od 0 do stElementov-1. Začetna vrednost parametra *trenutnaGlobina* mora biti 0.

Algoritem na sodih globinah najprej poišče vozlišče številka $stKorena$, ki bo shranjeno v korenu trenutnega poddrevesa. Vozlišča, ki so v seznamu $vozlUrPoX$ pred njim, sodijo v levo poddrevo, tista za njim pa v desno. Skozi seznam $vozlUrPoY$ se je treba sprehoditi in za vsako vozlišče v seznamu odločiti, ali sodi v levo poddrevo (v tem primeru ga dodamo v lokalni seznam $vozlUrLeva$), desno poddrevo (lokalni seznam $vozlUrDesna$) ali v koren, s čimer ohranimo urejenost obeh seznamov. S pomočjo obeh lokalnih seznamov lahko rekurzivno zgradimo obe poddrevesi. Na lihih globinah je postopek enak, le pomen koordinat x in y je zamenjan [FBF77, BKOS00].

```
ALGORITEM zgradiKDDrevo(koren, vozlUrPoX, vozlUrPoY, trenutnaGlobina)
 1  vozlUrLeva, vozlUrDesna//seznama vozlišč, namenjenih za obe poddrevesi, urejena po
                            koordinati, po kateri se vozlišča NE delijo na trenutni globini
 2  koren.globina ← trenutnaGlobina
 3  koren.podrocje.xMin ← vozlUrPoX[0].x
 4  koren.podrocje.xMax ← vozlUrPoX[ZADNJI].x
 5  koren.podrocje.yMin ← vozlUrPoY[0].y
 6  koren.podrocje.yMax ← vozlUrPoY[ZADNJI].y

 7  if trenutnaGlobina mod 2 = 0

 8     then //na sodi globini vozlišča delimo po x-koordinati
 9          stKorena ← ⌊koren.stVozlisc/2⌋
10          while stKorena > 0 ∧ vozlUrPoX[stKorena].x = vozlUrPoX[stKorena − 1].x
11          do stKorena ← stKorena − 1
12          koren.{x, y, indeks, robni} ← vozlUrPoX[stKorena].{x, y, indeks, robni}
13          for v in vozlUrPoY
14          do if v.indeks ≠ koren.indeks ∧ v.x < koren.x
15                then vozlUrLeva.DODAJ(v)
16             if v.indeks ≠ koren.indeks ∧ v.x >= koren.x
17                then vozlUrDesna.DODAJ(v)
18          if vozlUrLeva.stElementov > 0
19            then koren.levi = NEW
20                 zgradiKDDrevo(koren.levi, vozlUrPoX[0 : stKorena − 1],
21                               vozlUrLeva, trenutnaGlobina + 1)
22          if vozlUrDesna.stElementov > 0
23            then koren.desni = NEW
24                 zgradiKDDrevo(koren.desni, vozlUrPoX[stKorena + 1 : ZADNJI],
25                               vozlUrDesna, trenutnaGlobina + 1)

26     else //enak postopek kot na sodi globini, vendar vozlišča delimo po y-koordinati
27          stKorena ← ⌊koren.stVozlisc/2⌋
28          while stKorena > 0 ∧ vozlUrPoY[stKorena].y = vozlUrPoY[stKorena − 1].y
29          do stKorena ← stKorena − 1
30          koren.{x, y, indeks, robni} ← vozlUrPoY[stKorena].{x, y, indeks, robni}
31          for v in vozlUrPoX
32          do if v.indeks ≠ koren.indeks ∧ v.y < koren.y
33                then vozlUrLeva.DODAJ(v)
34             if v.indeks ≠ koren.indeks ∧ v.y >= koren.y
35                then vozlUrDesna.DODAJ(v)
36          if vozlUrLeva.stElementov > 0
37            then koren.levi = NEW
38                 zgradiKDDrevo(koren.levi, vozlUrLeva,
39                               vozlUrPoY[0 : stKorena − 1], trenutnaGlobina + 1)
40          if vozlUrDesna.stElementov > 0
41            then koren.desni = NEW
42                 zgradiKDDrevo(koren.desni, vozlUrDesna,
43                           vozlUrPoY[stKorena + 1 : ZADNJI], trenutnaGlobina + 1)
```

Z ZADNJI smo označili indeks zadnjega elementa v seznamu, s $seznam[i : j]$ pa vse elemente od vključno i-tega do vključno j-tega. Časovna zahtevnost zanke v vrsticah 10-11 je navadno konstantna, v izrojenem primeru pa $O(N)$. Zanka v vrsticah 14-17 se v vsakem primeru izvaja $O(N)$ časa. Sledita še dva rekurzivna klica, katerih parametri so seznami polovičnih dolžin, torej gre za algoritem vrste *deli in vladaj*. Čas izvajanja prikazanega algoritma ZGRADIKDDREVO z N vozlišči izrazimo kot [CLRS01]:

$$T(N) = c \cdot N + 2T(N/2).$$

Iz [CLRS01] vemo, da je asimptotična rešitev te rekurzivne enačbe $O(N \log N)$. Enaka je tudi zahtevnost predhodnega urejanja vozlišč, torej je tudi skupna asimptotična časovna zahtevnost gradnje k-D drevesa enaka $O(N \log N)$.

Iskanje vozlišč znotraj danega kroga

Zgrajeno drevo nam omogoča učinkovito iskanje vozlišč znotraj poljubnega danega lika [FBF77]. Vhodni podatki uporabljenega algoritma so: *koren* k-D drevesa, središče kroga $\mathbf{x}_C$ in premer kroga d. Izhodni parameter je *seznamVozlisc*, v katerega naj se shranijo najdena vozlišča, ki mora biti na začetku prazen. Algoritem uporablja iskanje v globino in ne zagotavlja nobene urejenosti vozlišč v končnem seznamu [BKOS00].

```
ALGORITEM NAJDIVOZLVKROGU(koren, x_C, d, seznamVozlisc)
1  if RAZDALJA(x_C, koren) < d/2
2     then seznamVozlisc.DODAJ(koren)
3  if koren.levi ≠ NULL ∧ RAZDALJA(x_C, koren.levi.podrocje) < d/2
4     then NAJDIVOZLVKROGU(koren.levi, x_C, d, seznamVozlisc)
5  if koren.desni ≠ NULL ∧ RAZDALJA(x_C, koren.desni.podrocje) < d/2
6     then NAJDIVOZLVKROGU(koren.desni, x_C, d, seznamVozlisc)
```

Funkcija RAZDALJA($\mathbf{t}_1, \mathbf{t}_2$) vrne razdaljo med točkama, RAZDALJA($\mathbf{t}, pravokotnik$) pa razdaljo od točke $\mathbf{t}$ do njej najbližje točke znotraj pravokotnika. Pogoja v vrsticah 3 in 5 sta izpolnjena, če krog seka področje levega oziroma desnega poddrevesa.

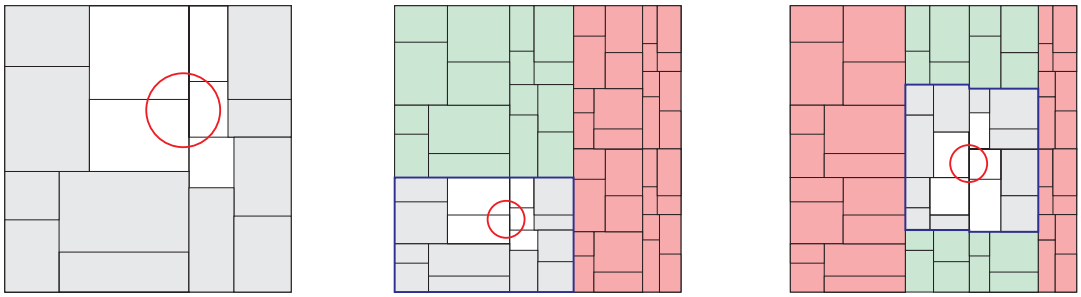
Zahtevnost iskanja

Izrojen primer: predpostavimo N vozlišč, razporejenih na enotsko krožnico s središčem v $(0, 0)$ in polmerom 1. Potem obstaja takšen $\epsilon > 0$, da za poljubni dve vozlišči njun očrtan pravokotnik seka krog s polmerom $1 - \epsilon$. Iskanje vozlišč v krogu s polmerom $1-\epsilon$ bo pomenilo preiskovanje vseh N drevesnih vozlišč, čeprav bo rezultat prazen seznam. Časovna zahtevnost je torej v najslabšem primeru enaka $O(N)$, vendar do tako izrojene razporeditve vozlišč v MPM ne more priti.

Iskanje v splošnem pravokotniku: navajeni smo, da iskanje v drevesnih podatkovnih strukturah zahteva $O(\log N)$ časa. Zahtevnost iskanja vozlišč znotraj poljubnega pravokotnika z uporabo k-D drevesa je večja – $O(\sqrt{N} + p)$, kjer je p število najdenih vozlišč [BKOS00], kar lahko ponazorimo s primerom. Denimo, da je N vozlišč približno enakomerno raztresenih po enotskem kvadratu tako, da je minimalna razdalja med njimi d_{min}. Iščemo vozlišča, ki ležijo znotraj nekega pravokotnika višine 1 in širine $\epsilon < d_{min}$. V k-D drevesu je $O(N)$ (natančneje vsaj $N/7$) takšnih poddreves, ki imajo 4-7 vozlišč. Očrtani pravokotniki vseh takšnih poddreves so široki vsaj ϵ, $O(\sqrt{N})$ izmed njih pa seka dani pravokotnik in jih je potrebno preiskati. Časovna zahtevnost je zato sorazmerna $\sqrt{N}$, četudi pri zelo majhnem ϵ pravokotnik ne vsebuje nobenega vozlišča.

Iskanje v krogu v primeru MPM: v našem primeru iščemo vozlišča znotraj kroga, kar je asimptotično enako zahtevno kot iskanje v kvadratu, očrtanem temu krogu. Poleg tega velikost kroga pada s številom vozlišč tako, da povprečno število $\overline{p}$ najdenih vozlišč ostaja konstantno. Za takšen primer lahko pričakujemo skromnejšo zahtevnost iskanja – $O(\log N + p)$, kar bomo neformalno prikazali na primeru.

Imejmo N vozlišč v enotskem kvadratu in pripadajoče k-D drevo. Očrtani pravokotniki poddreves na neki globini so prikazani na levi polovici slike 5.3. Pri iskanju znotraj rdečega kroga lahko najpozneje na tej globini zavržemo vsa poddrevesa, katerih očrtani pravokotniki so obarvani sivo. Označimo čas takšnega iskanja s $T(N)$.



Slika 5.3: Iskanje vozlišč, ki ležijo v danem krogu, v osnovnem $k-$D drevesu (levo) in dva primera iskanja v krogu s štirikrat manjšo ploščino v drevesu s štirikrat večjim številom vozlišč (sredina in desno).

Početverimo število vozlišč, da dobimo drevo, ki je na enak način prikazano na srednjem delu slike 5.3. Štirikratno število vozlišč v MPM pomeni štirikrat manjšo ploščino iskalnega kroga. V prvem koraku iskanja na globini 0 zavržemo desno poddrevo, na sliki obarvano rožnato, v drugem koraku na globini 1 pa zgornje poddrevo, obarvano zeleno. Preostanek iskanja poteka v modro očrtanem pravokotniku in je enako iskanju med N vozlišči. Skupaj iskanje traja:

$$T(4N) = c + T(N) \tag{5.2}$$

časa.

Podobno bi veljalo, če bi bil iskalni krog drugje (slika 5.3 desno). Čeprav bi morali v tem primeru na globinah 0 in 1 iskanje nadaljevati po vseh vejah, bi na globini 2 zavrgli rožnato obarvani del in na globini 3 zelenega, nakar bi bilo število operacij do konca iskanja enako, kot pri prvotnem številu vozlišč. Spet velja rekurzivna enačba 5.2, katere asimptotična rešitev je $O(\log N)$, saj početverjenje števila vozlišč iskanje podaljša le za konstantno število operacij.
Sklep: Pričakovana zahtevnost iskanja vozlišč v krogu, katerega ploščina pada z N tako, da je povprečno število $\overline{p}$ najdenih vozlišč konstantno, je $O(\log N + \overline{p})$.

Določanje premera nosilne domene

Opisali bomo različico zAvgId, ki zveznost poskusne funkcije vsiljuje z interpolacijo povprečne razdalje $\overline{d}$. Najprej je treba ovrednotiti $\overline{d}$ na pravilni mreži velikosti $l \times l$. Vhodni podatki so *koren* k-D drevesa z vozlišči, velikost pravilne mreže l, parameter α_S in povprečna razdaja med vozlišči v celotni domeni $\overline{d}^* = \sqrt{A_\Omega}/(\sqrt{N} - 1)$, kjer je A_Ω ploščina celotne domene, na kateri rešujemo PDE. Rezultat funkcije je matrika $\overline{D}$.

ALGORITEM OVREDNOTIPOVPRAZD($koren, l, \alpha_S, \overline{d}^*$)
1 $n_I \leftarrow$ po enačbi 4.3 na strani 71
2 **for** $ix \leftarrow 1$ **to** l
3 **do for** $iy \leftarrow 1$ **to** l
4 **do** $\mathbf{x} \leftarrow$ točka, ki ustreza (ix, iy) na reg. mreži
5 $D_S \leftarrow$ ocena premera vzorčne domene
6 **repeat**
7 $Z \leftarrow prazenSeznam$//vozlišča v vzorčni domeni
8 NAJDIVOZLVKROGU($koren, \mathbf{x}, D_S, Z$)
9 $D_S \leftarrow 1.5 \cdot D_S$
10 **until** Z.STELEMENTOV $< n_I$
11 uredi Z po naraščajoči razdalji od $\mathbf{x}$
12 $\overline{D}[ix, iy] \leftarrow \frac{\sqrt{\pi}}{2\sqrt{n_I}-1} \cdot$ (RAZDALJA($Z[n_I - 1], \mathbf{x}$) + RAZDALJA($Z[n_I], \mathbf{x}$))

Oceno D_S v vrstici 5 lahko dobimo iz $\overline{d}^*$ in parametra α_S. Zanka 6-10 je potrebna, ker k-D drevo ne podpira neposrednega iskanja najbližjih n_I sosedov [BKOS00]. Zanka najpozneje v nekaj prehodih najde dovolj vozlišč, vsak prehod pa traja $O(\log N + n_I)$ časa. Urejanje doda še $O(n_I \log n_I)$, zato je skupna zahtevnost funkcije OVREDNOTIPOVPRAZD $O(l^2(\log N + n_I \log n_I))$. Eksperimentalno smo ugotovili, da za dobro oceno povprečnih razdalj pri 400 in več vozliščih povsem zadošča mreža velikosti $l = \sqrt{N}/5$. Čas za ovrednotenje $\overline{d}$ na mreži takšne velikosti je v primerjavi s skupno zahtevnostjo MPM zanemarljiv, zato nismo raziskovali vpliva še redkejše mreže. Velikosti d_S nosilne in d_Q kvadraturne domene točke $\mathbf{x}$ dobimo po enačbi 3.13 na strani 49.

Podatkovne strukture za spremenljive domene

V primeru, da se domena s časom spreminja, se spreminjajo tudi koordinate vozlišč. Uporabiti je treba *kinetično* podatkovno strukturo, to je takšno, ki podpira premika-

nje vozlišč v prostoru in temu ustrezno prilagajanje *kombinatorične strukture podatkov* [AGG02]. Dasiravno ni znana nobena takšna različica k-D dreves, obstajajo podatkovne strukture, zasnovane na *razdelitvenih drevesih* (angl. partition trees) in *R-drevesih* [AAE03, KGT99], ki so primerne tudi za spreminjajoče se domene. Nekatere izmed njih odlikuje hitro iskanje (celo v času $O(\log N + p)$ za splošni primer) na račun zahtevnejšega premikanja vozlišč, druge pa obratno. Vsem, ki so pri iskanju hitrejše od $O(\sqrt{N} + p)$ za splošni primer, je skupna zapletenost, zato je njihova praktična uporabnost precej omejena [AGG02].

5.3.2 Računanje vrednosti baznih funkcij

V dani točki $\mathbf{x}$ imajo neničelne funkcijske vrednosti in odvode natanko tiste bazne funkcije, ki imajo vrhove v nosilnih vozliščih točke $\mathbf{x}$. Vhodni podatki funkcije za izračun vrednosti baznih funkcij in odvodov so torej $\mathbf{x}$ in podatki za določanje njene nosilne domene: *koren* k-D drevesa z vozlišči, parameter α_S in matrika povprečnih razdalj $\overline{D}$. Rezultat so vektorji vrednosti baznih funkcij ϕ in po potrebi njenih odvodov ϕ_x, ϕ_y ter seznam I indeksov pripadajočih vozlišč. Dolžine vektorjev so enake številu nosilnih vozlišč.

ALGORITEM BAZNEFUNKCIJE($\mathbf{x}, koren, \alpha_S, \overline{D}, izracunajOdvode$)

1 $d_S \leftarrow \alpha_S \cdot$ INTERPOLIRAJ($\overline{D}, \mathbf{x}$)
2 $S \leftarrow prazenSeznam$//nosilna vozlišča točke $\mathbf{x}$
3 NAJDIVOZLVKROGU($koren, \mathbf{x}, d_S, S$)
4 $\{A, A_x, A_y\} \leftarrow 0$//matrike velikosti $m \times m$
5 **for** $k \leftarrow 0$ **to** S.STELEMENTOV $- 1, \mathbf{x}_k \leftarrow S[k]$
6 **do for** $i \leftarrow 1$ **to** m
7 **do for** $j \leftarrow 1$ **to** m
8 **do** $A[i,j] \leftarrow A[i,j] + W_{MLS}(\mathbf{x}, \mathbf{x}_k, d_S) p_i(\mathbf{x}_k) p_j(\mathbf{x}_k)$
9 **if** $izracunajOdvode$
10 **then** $A_x[i,j] \leftarrow A_x[i,j] + W_{MLS,x}(\mathbf{x}, \mathbf{x}_k, d_S) p_i(\mathbf{x}_k) p_j(\mathbf{x}_k)$
11 $A_y[i,j] \leftarrow A_y[i,j] + W_{MLS,y}(\mathbf{x}, \mathbf{x}_k, d_S) p_i(\mathbf{x}_k) p_j(\mathbf{x}_k)$
12 $\{L, U, P\} \leftarrow$ RAZCEPILU(A)//razcepi $A = L \cdot U \cdot P$, P je permutacijska matrika
13 **if** $izracunajOdvode$
14 **then** $\mathbf{g} \leftarrow$ RESILU($L, U, P, \mathbf{p}(\mathbf{x})$)
15 $\mathbf{d}_x \leftarrow \mathbf{p}_x^{\mathrm{T}}(\mathbf{x}) - \mathbf{g}^{\mathrm{T}} A_x$
16 $\mathbf{d}_y \leftarrow \mathbf{p}_y^{\mathrm{T}}(\mathbf{x}) - \mathbf{g}^{\mathrm{T}} A_y$
17 **for** $j \leftarrow 0$ **to** S.STELEMENTOV $- 1, \mathbf{x}_j \leftarrow S[j]$
18 **do** $\mathbf{a} \leftarrow$ RESILU($L, U, P, W_{MLS}(\mathbf{x}, \mathbf{x}_j, d_S)\mathbf{p}(\mathbf{x}_j)$) // izračunaj j-ti stolpec matrike $A^{-1}B$
19 $I[j] \leftarrow \mathbf{x}_j.index$
20 $\phi[j] \leftarrow \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{a}$
21 **if** $izracunajOdvode$
22 **then** $\phi_x[j] \leftarrow \mathbf{d}_x\mathbf{a} + \mathbf{g}^{\mathrm{T}} W_{MLS,x}(\mathbf{x}, \mathbf{x}_j, d_S)\mathbf{p}(\mathbf{x}_j)$
23 $\phi_y[j] \leftarrow \mathbf{d}_y\mathbf{a} + \mathbf{g}^{\mathrm{T}} W_{MLS,y}(\mathbf{x}, \mathbf{x}_j, d_S)\mathbf{p}(\mathbf{x}_j)$

Z $W_{MLS}(\mathbf{x}_1, \mathbf{x}_2, d_S)$ smo označili utežno funkcijo za aproksimacijo MLS, podano v enačbi 3.4 na strani 45. Zanka 5-11 po enačbi 3.7 izračuna matriko A in po potrebi njena

odvoda. Sledi LU razcep A. Po potrebi rešimo sistem $A\mathbf{g} = \mathbf{p}(\mathbf{x})$ (enačba 3.10) in izračunamo vsebino oklepaja v zadnji vrstici enačbe 3.11. V zanki 17-23 izračunamo vrednosti posamičnih baznih funkcij in po potrebi njihovih odvodov. Najprej izračunamo ustrezen stolpec rešitve sistema $A^{-1}B$, ki ga uporabimo v enačbah 3.8 in 3.11.

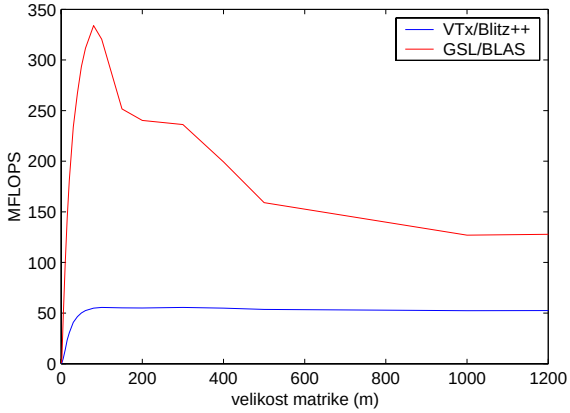
Preizkusili smo dve knjižnici, ki ponujata funkciji za LU razcep (vrstica 12) in reševanje dobljenih trikotnih sistemov (vrstici 14 in 18):

- na *šablonskih izrazih* (angl. template expressions) [VJ97] zasnovani Blitz++ [Vel] z dodatkom _VTx [Sur]. Uporaba šablonskih izrazov naj bi omogočila prevajalniku kakovostno optimizacijo. Knjižnica naj bi tudi preurejala vrstni red operacij za čimvečjo verjetnost predpomnilniškega zadetka (angl. se postopek imenuje iteration-space tiling). [Wol87]
- V čistem jeziku C napisana GNU Scientific Library (GSL) [GDT$^+$], ki uporablja BLAS (Basic Linear Algebra Subprograms) [LHKK79, BLA], de facto standardno knjižnico za osnovne vektorske in matrične operacije.

LU razcep polne matrike zahteva približno $N^3/3$ operacij s plavajočo vejico [Hea02]. Nobena od knjižnic ne izvaja nepotrebnih izračunov v plavajoči vejici, zato lahko njuno hitrost izrazimo kar v milijonih operacij s plavajočo vejico na sekundo (angl. million of floating point operations per second, MFLOPS). Hitrejša knjižnica doseže več operacij v sekundi in se tako bolj približa teoretični zmogljivosti računalnika. Počasnejša knjižnica mora ob enaki velikosti matrike opraviti enako število operacij, a jih opravi manj v sekundi, zato razcep traja dlje.

Hitrost obeh v odvisnosti od velikosti matrike m smo izmerili na testnem računalniku s procesorjem AMD opteron 244 s frekvenco 1.8 GHz, 128 KB prvonivojskega podatkovnega predpomnilnika in 1 MB drugonivojskega skupnega predpomnilnika. Na računalniku je tekel operacijski sistem Fedora Core 2 linux z jedrom 2.6.8-1.521smp in prevajalnik `gcc` verzije 3.3.3. Hitrost funkcije za LU razcep v obeh knjižnicah je podana na grafu 5.4. Pri GSL sta očitna padca hitrosti pri $m \approx 120$, ko podatki postanejo preveliki za prvonivojski predpomnilnik, in pri $m \approx 400$, ko je tudi drugonivojski predpomnilnik premajhen. Blitz++ vrstni red uspešno prilagaja in povsem odpravi odvisnost od velikosti predpomnilnika, vendar je vseskozi počasnejši. V izračunu vrednosti baznih funkcij MPM pravzaprav nastopajo le matrike velikosti 3×3 (pri MPM1) in 6×6 (pri MPM2). Pri teh velikostih je razmerje hitrosti okrog 8 v korist GSL, ki smo jo zato izbrali za našo izvedbo.

Kot smo pokazali, je zahtevnost iskanja vozlišč S v nosilni domeni enaka $O(\log N + |S|)$. Postopek zAvgId za določanje velikosti nosilne domene je dovolj dober, da v povprečju velja $|S| = n_I$, slednjega pa s parametrom α_S povezuje enačba 4.3. Inicializacija matrik v vrstici 4 zahteva $O(m^2)$ časa. Sledi zanka za gradnjo matrik A, A_x in A_y (5-11), ki traja $O(|S|m^2)$. Vektorje $\mathbf{p}(\mathbf{x})$, $\mathbf{p}(\mathbf{x}_j)$, vrednosti W_{MLS} in njihove odvode se izplača izračunati vnaprej, saj jih uporabimo večkrat.



Slika 5.4: Hitrost funkcije za LU razcep matrike velikosti $m \times m$ v knjižnicah Blitz++ in GSL.

Časovna zahtevnost LU razcepa je $O(m^3)$, vsakega reševanje dobljenega trikotnega sistema pa $O(m^2)$. Prevladujoča operacija v vrsticah 15-16 je množenje vektorja z matriko, za kar potrebujemo $O(m^2)$ operacij. Končna zanka v vrsticah 17-23 vsebuje reševanje trikotnega sistema in nekaj manj zahtevnih operacij.

Skupna časovna zahtevnost funkcije BAZNEFUNKCIJE je torej $O(\log N + n_I + m^2 + m^2 n_I + m^3 + m^2 + m^2 n_I)$, kar je z upoštevanjem $n_I \geq m$ enako $O(\log N + m^2 n_I)$. Če bi za MPM1 in MPM2 uporabili enako število kvadraturnih točk, bi se njuni zahtevnosti razlikovali zgolj v tej funkciji, saj m ne nastopa nikjer drugje.

5.3.3 Gaussova integracija

Zadnji gradnik MPM je izračun integralov K^{int} in C^{int} iz enačbe 3.22. Glede na ugotovitve, podane v poglavju 4.5.3, izberemo ustrezne abscise in uteži kvadraturnih točk. Ker je kvadraturna domena pravokotnik, so kvadraturne točke za dvodimenzionalno integracijo kartezični produkt abscis za enodimenzionalno integracijo, uteži pa njihovi zmnožki [AS76]. Nato za vsako kvadraturno točko izračunamo vse njene prispevke k elementom matrik A in B. Prispevki za posamezno vrstico matrik torej niso razvrščeni po stolpcih oziroma vozliščih, marveč po kvadraturnih točkah, zato v funkciji SESTAVISISTEMMPM zanko po vozliščih v vrsticah 7-10 zamenjamo z zanko po kvadraturnih točkah.

ALGORITEM IZRACUNAJINTEGRALE($koren, \mathbf{x}_i, \Delta t, c, \alpha_S, \alpha_Q, \overline{D}$)

1 $\mathbf{x}_G, \mathbf{w}_G$// abscise z intervala (-1,1) in uteži za 1-D Gaussovo integracijo z n_G točkami
2 $d_Q \leftarrow \alpha_Q \cdot$ INTERPOLIRAJ($\overline{D}, \mathbf{x}_i$)
3 **for** $\mathbf{x}_Q \in (\mathbf{x}_G \frac{d_Q}{2} + \mathbf{x}_i.x) \times (\mathbf{x}_G \frac{d_Q}{2} + \mathbf{x}_i.y), w_Q \in \mathbf{w}_G \mathbf{w}_G^{\mathrm{T}} \cdot d_Q^2/4$
4 **do** // $\mathbf{x}_Q$ je ena izmed kvadraturnih točk 2-D Gaussove integr., w_Q pa pripadajoča utež
5 $\{\phi, \phi_x, \phi_y, I\} \leftarrow$ BAZNEFUNKCIJE($\mathbf{x}_Q, koren, \alpha_S, \overline{D}, true$)
6 **for** $k \leftarrow 0$ **to** ϕ.STELEMENTOV $- 1$

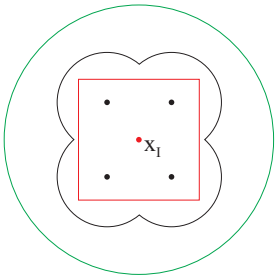
7 **do** $K^{int} \leftarrow w_Q \cdot c \cdot [W_{MLPG,x}(\mathbf{x}_i, \mathbf{x}_Q, d_Q)\phi_x[k] + W_{MLPG,y}(\mathbf{x}_i, \mathbf{x}_Q, d_Q)\phi_y[k]]$
8 $C^{int} \leftarrow w_Q \cdot W_{MLPG}(\mathbf{x}_i, \mathbf{x}_Q, d_Q)\phi[k]$
9 $seznamA$.DODAJ($\mathbf{x}_i.indeks, I[k], 2C^{int} + \Delta t K^{int}$)
10 $seznamB$.DODAJ($\mathbf{x}_i.indeks, I[k], 2C^{int} - \Delta t K^{int}$)
11 **for** $\mathbf{x}_Q \in$ kvadraturne točke, razporejene po vsakem odseku $\Gamma_{Q,bound} = \Omega_Q \cap \Gamma$
12 **do** $\mathbf{w}_Q \leftarrow$ pripadajoča utež kvadraturne točke $\mathbf{x}_Q$, pomnožena s polovico dolžine odseka
13 $\{\phi, \phi_x, \phi_y, I\} \leftarrow$ BAZNEFUNKCIJE($\mathbf{x}_Q, koren, \alpha_S, \overline{D}, true$)
14 **for** $k \leftarrow 0$ **to** ϕ.STELEMENTOV $- 1$
15 **do** $K^{int} \leftarrow -w_Q \cdot c \cdot W_{MLPG}(\mathbf{x}_i, \mathbf{x}_Q, d_Q)(\phi_x[k] \cdot n_x + \phi_y[k] \cdot n_y)$
16 $seznamA$.DODAJ($\mathbf{x}_i.indeks, I[k], \Delta t K^{int}$)
17 $seznamB$.DODAJ($\mathbf{x}_i.indeks, I[k], -\Delta t K^{int}$)

Z W_{MLPG} smo označili testno funkcijo, podano v enačbi 3.16 na strani 55.

Zanka v vrsticah 3-10 se izvede n_G^2-krat. V vsakem obhodu izračun vrednosti baznih funkcij in njihovih odvodov traja $O(\log N + m^2 n_I)$ časa, preostanek zanke pa $O(n_I)$. Zanka v vrsticah 11-17 se za vsak odsek $\Gamma_{Q,bound}$ roba domene Ω, ki ga seka kvadraturna domena Ω_Q, izvede n_G-krat in ima enako zahtevnost na obhod kot prva zanka. Število N_Γ notranjih vozlišč, katerih kvadraturna domena seka rob domene Ω, je sorazmerno $\sqrt{N}$, zato asimptotično druga zanka nima vpliva na skupno zahtevnost, ki je potemtakem $O(n_G^2(\log N + m^2 n_I))$.

Število vozlišč N_Γ je odvisno tudi od parametra α_Q. Pri za MPM1 optimalnem izboru $\alpha_Q = 1.4$ smo našteli okrog $2.5\sqrt{N}$ takšnih vozlišč, pri za MPM2 optimalni vrednosti $\alpha_Q = 0.8$ pa nobenega, ker so z našim postopkom dobljena vozlišča vsaj za predpisano minimalno razdaljo oddaljena od roba. Število ni odvisno od lokalne gostote vozlišč, saj zgostitev vozlišč v nekem delu domene Ω in s tem zmanjšanje povprečne razdalje $\overline{d}$ med vozlišči povzroči zmanjšanje kvadraturnih domen v tem delu.

Preizkusili smo tudi različico iščiKvad (osnovno različico smo poimenovali iščiOsn), ki najprej najde vsa vozlišča v nosilni domeni Ω_{SQ} kvadraturne domene Ω_Q, nato pa vsakič išče zgolj med njimi. Primer odnosa med domenami kaže slika 5.5, kjer sta vozlišče $\mathbf{x}_I$ in njegova kvadraturna domena narisana rdeče. Prvo iskanje izvedemo v krogu, očrtanem



Slika 5.5: Kvadraturna domena Ω_Q vozlišča $\mathbf{x}_I$ (rdeče), krog, očrtan nosilni domeni Ω_{SQ} kvadraturne domene vozlišča (zeleno) in unija nosilnih domen vseh štirih kvadraturnih točk (črno).

Ω_{SQ}, ki ima premer $d_{SQ} = (\alpha_S + \sqrt{2} \cdot \alpha_Q)\overline{d}$ (zeleni krog na sliki). Njegova ploščina je za faktor

$$f = \left(\frac{d_{SQ}}{d_S}\right)^2 = \left(\frac{\alpha_S + \sqrt{2} \cdot \alpha_Q}{\alpha_S}\right)^2$$

večja od ploščine nosilne domene posamezne točke in za isti faktor se v povprečju poveča tudi število najdenih vozlišč. V primeru za MPM1 optimalnih parametrov je $f_{MPM1} \approx 3.6$, za MPM2 pa $f_{MPM2} \approx 2.1$. Vozlišča v uniji nosilnih domen vseh kvadraturnih točk (črno obrobljeno področje) bomo dejansko uporabili za integracijo, vozlišč iz preostanka zelenega kroga pa ne.

Med približno $f \cdot n_I$ vozlišči, najdenimi v razširjenem krogu, vsakokrat iščemo s *popolnim prebiranjem*. Njihovo število je namreč premajhno, da bi se izplačalo graditi novo majhno k-D drevo. Za iskanje vozlišč v Ω_{SQ} potrebujemo $O(\log N + fn_I)$ časa, nato pa v vsaki kvadraturni točki $O(fn_I)$ za iskanje nosilnih vozlišč in $O(m^2 n_I)$ za izračun vrednosti baznih funkcij in njihovih odvodov. Skupna zahtevnost različice iščiKvad je torej $O(\log N + n_G^2 n_I(f + m^2))$.

5.3.4 Zahtevnost celotnega postopka

Z vsemi opisanimi gradniki lahko povzamemo celoten postopek sestavljanja linearnega sistema z MPM. Najprej je potrebno dana vozlišča urediti po x- in y-koordinati ter iz njih zgraditi k-D drevo, nato pa ovrednotiti povprečno razdaljo na pravilni mreži. Sledi računanje prispevkov k elementom sistemskih matrik in na koncu njihovo sestavljanje.

```
ALGORITEM SESTAVISISTEMMPM(V, Δt, c, α_S, α_Q, A_Ω)
 1  koren ← NEW
 2  ZGRADIKDDREVO(koren, UREDIPOX(V), UREDIPOY(V), 0)
 3  D̄ ← OVREDNOTIPOVPRAZD(koren, max(√N/5, 4), α_S, √A_Ω/(√N − 1) )
 4  for each x_i ∈ V
 5  do seznamA, seznamB//v vsakem obhodu na začetku prazna seznama, v katera dodajamo
                         indekse in vrednosti neničelnih elementov matrik A in B
 6     if vozlišče x_i je robno
 7        then {φ, I} ← BAZNEFUNKCIJE(x_i, koren, α_S, D̄, false)
 8             for k ← 0 to φ.STELEMENTOV − 1
 9             do seznamA.DODAJ(x_i.indeks, I[k], φ[k])
10        else IZRACUNAJINTEGRALE(koren, x_i, Δt, c, α_S, α_Q, D̄)
11     A ← SESTAVIREDKOMATRIKO(seznamA, |V|)
12     B ← SESTAVIREDKOMATRIKO(seznamB, |V|)
```

Izračunane elemente dodajamo v matriki po vsakem obhodu zanke posebej, saj vsak obhod pomeni eno vrstico matrik s povprečno $n_G^2 n_I$ prispevki. Če elemente seznamov pred vsakim dodajanjem uredimo po številki stolpca, je skupna asimptotična zahtevnost urejanja in dodajanja urejenega seznama v matriko enaka $O(n_G^2 n_I \log(n_G^2 n_I))$.

Časovna zahtevnost

Skupna zahtevnost z različico iščiOsn algoritma IZRACUNAJINTEGRALE je tako:

$$\begin{aligned} T_{MPM} = O(N \cdot [\quad & \log N + && \text{(vrstica 2)} \\ & \log N + n_I \log n_I + && \text{(vrstica 3)} \\ & n_G^2 \log N + n_G^2 m^2 n_I + && \text{(vrstice 6-10)} \\ & n_G^2 n_I \log(n_G^2 n_I) \ ] \ ) && \text{(vrstici 11-12)} \\ \approx O(N \cdot [\quad & n_G^2 \log N + n_I \log n_I + n_G^2 m^2 n_I \ ]), \end{aligned}$$

kjer smo upoštevali, da velja $\log(n_G^2 n_I) < m^2$.

Skupna zahtevnost različice iščiKvad je:

$$\begin{aligned} T_{MPM}^{isciKvad} = O(N \cdot [\quad & \log N + && \text{(vrstica 2)} \\ & \log N + n_I \log n_I + && \text{(vrstica 3)} \\ & \log N + n_G^2 n_I f + n_G^2 m^2 n_I + && \text{(vrstice 6-10)} \\ & n_G^2 n_I \log(n_G^2 n_I) \ ] \ ) && \text{(vrstici 11-12)} \\ \approx O(N \cdot [\quad & \log N + n_I \log n_I + n_G^2 m^2 n_I \ ]), \end{aligned}$$

kjer smo upoštevali, da velja $f < m^2$.

Prostorska zahtevnost

Seznam V, urejen po x-koordinati, zahteva $3N \cdot e$ prostora, enako kot njegova po y-koordinati urejena kopija. Vsako izmed N drevesnih vozlišč s strukturo, navedeno na strani 90, potrebuje $10e$ prostora. Med gradnjo drevesa imata na vsaki globini g lokalna seznama *vozlUrLeva* in *vozlUrDesna* skupaj $N/2^g$ vozlišč. Zaradi rekurzivne narave gradnje hkrati hranimo sezname na vseh globinah, ki vsebujejo skupaj $2N$ vozlišč, kar pomeni $2 \cdot 3N \cdot e$ prostora. Prostorska zahtevnost vrstice 2 je torej $(2 \cdot 3 + 10 + 2 \cdot 3)N \cdot e = 22N \cdot e$, nakar lahko zavržemo vse spremenljivke razen zgrajenega drevesa velikosti $10N \cdot e$.

Matrika $\overline{D}$ porabi $N/25 \cdot e$ prostora. Vsak izmed seznamov za gradnjo matrik v vsakem obhodu zanke v vrsticah 4-10 pridobi $n_G^2 n_I$ elementov, vendar na koncu vsakega obhoda prispevke iz seznamov prištejemo k matrikam in seznama zavržemo, zato je njun vpliv na skupno prostorsko zahtevnost zanemarljiv. Velikost ostalih spremenljivk v zanki in klicanih funkcijah prav tako ni odvisna od N. Vsaka izmed končnih matrik ima nekaj manj kot $Nn_I f$ neničelnih elementov. Skupna prostorska zahtevnost je:

$$\begin{aligned} S_{MPM} &\approx \max(22, 10 + 1/25 + 3fn_I)N \cdot e \\ &\approx (10 + 3fn_I)N \cdot e \\ &\approx 300N \text{ do } 2000N \text{ bajtov.} \end{aligned}$$

5.4 Eksperimentalna analiza zahtevnosti

Asimptotična časovna zahtevnost je pregrob podatek za primerjavo metod, saj ne poznamo niti multiplikativne konstante niti deležev časa po posameznih korakih algoritma. Za podrobno eksperimentalno analizo smo zato MPM razdelili na naslednje vsebinske gradnike:

1. gradnja k-D drevesa,
2. ovrednotenje $\overline{d}$ na pravilni mreži,
3. iskanje vozlišč v nosilni domeni vsake kvadraturne točke,
4. računanje vrednosti baznih funkcij in njihovih odvodov v vsaki kvadraturni točki in sestavljanje seznamov prispevkov v matriki A in B,
5. končno sestavljanje matrik iz seznamov prispevkov.

Generiranju vozlišč za MPM nismo posvečali posebne pozornosti, ker traja zanemarljivo malo časa.

Najprej nas bo zanimalo, kolikšen delež skupnega časa si vzame posamezen gradnik, nato pa bomo analizirali vpliv parametrov na posamezne gradnike. Tabela 5.1 povzema asimptotične časovne zahtevnosti posameznih gradnikov in celotne MPM. Zahtevnosti iskanja vozlišč in računanja vrednosti baznih funkcij sta večji, kot smo navedli v prejšnjem razdelku, ker sta pomnoženi s skupnim številom ponovitev vsake operacije.

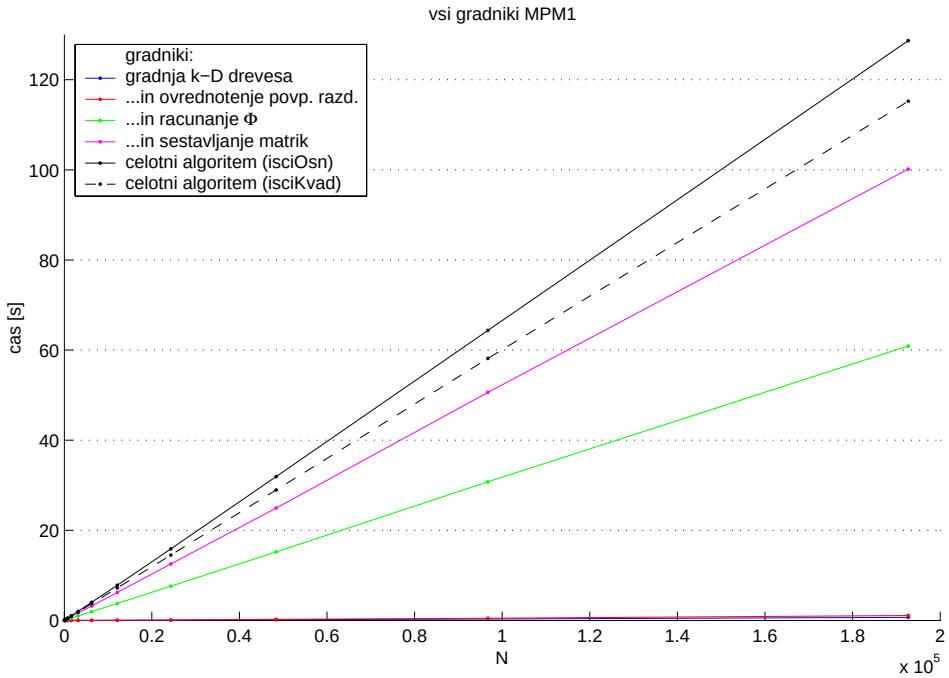
Vsi izmerjeni časi so dobljeni z optimizirano različico MPM, napisano v jeziku C++ in prevedeno s prevajalnikom `gcc` verzije 3.3.3. Uporabljali smo testne računalnike s procesorjem AMD opteron 244, opisane na strani 97.

Tabela 5.1: Asimptotična časovna zahtevnost posameznih gradnikov in celotne MPM.

gradnik	časovna zahtevnost
gradnja k-D drevesa	$N \log N$
ovrednotenje $\overline{d}$ na pravilni mreži	$N(\log N + n_I \log n_I)$
iskanje vozlišč v Ω_S	$N(n_G^2 \log N + n_G^2 n_I)$
računanje $\mathbf{\Phi}, \mathbf{\Phi}_x$ in $\mathbf{\Phi}_y$	$N n_G^2 m^2 n_I$
sestavljanje matrik A in B	$N n_G^2 n_I \log(n_G^2 n_I)$
celotna MPM	$N(n_G^2 \log N + n_G^2 m^2 n_I)$
iskanje vozlišč v Ω_S (različica iščiKvad)	$N(\log N + n_G^2 n_I f)$
celotna MPM (različica iščiKvad)	$N(\log N + n_G^2 m^2 n_I)$

5.4.1 Zahtevnost posameznih gradnikov MPM

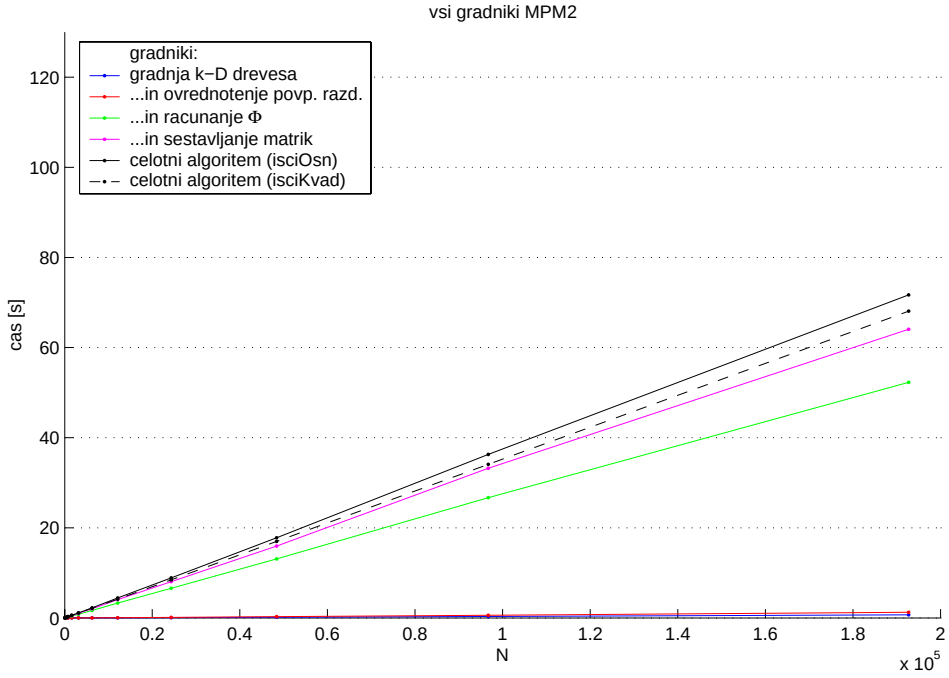
Slika 5.6 prikazuje čase, porabljene za posamezne gradnike MPM1 z optimalnimi parametri, naštetimi v tabeli 4.5.3. Modra črta predstavlja čas za gradnjo k-D drevesa, razlika od modre do rdeče čas za ovrednotenje $\overline{d}$ na pravilni mreži itd. Čas za iskanje vozlišč v nosilnih domenah kvadraturnih točk z različico iščiOsn predstavlja razlika od vijolične do polne črne črte, za različico iščiKvad pa od vijolične do črtkane črne črte.
Sklep: Gradnja drevesa in ovrednotenje $\overline{d}$ predstavljata zanemarljiv delež skupnega časa. Najzahtevnejša sta izračun vrednosti baznih funkcij in sestavljanje matrik. Iskanje vozlišč je z različico iščiKvad približno dvakrat hitrejše od osnovne različice, vendar predstavlja razmeroma majhen del skupnega časa.



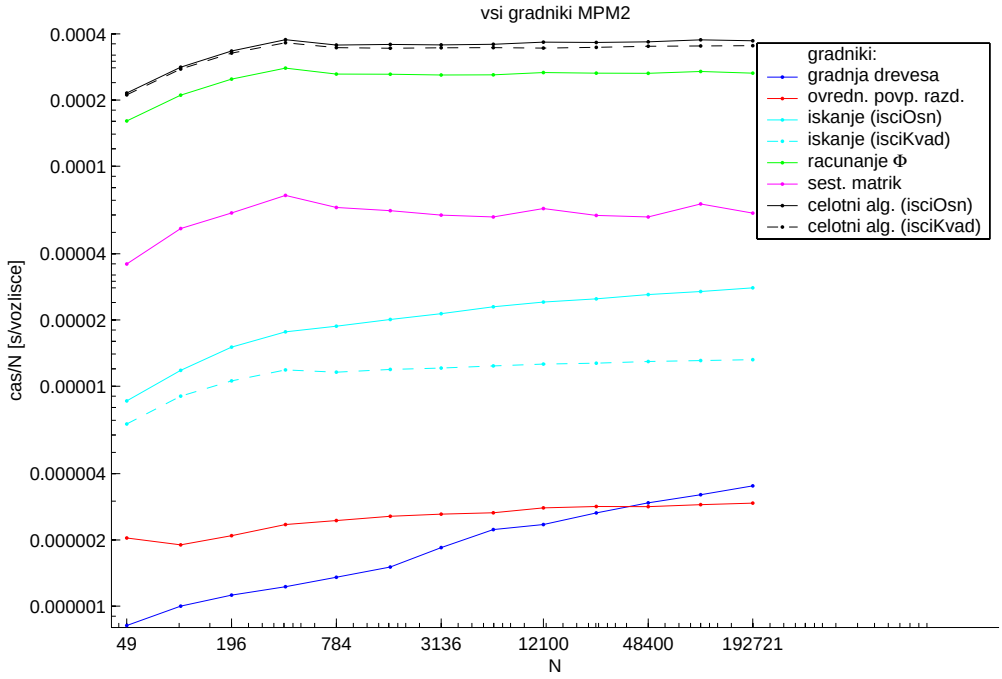
Slika 5.6: Čas, porabljen za MPM1 z optimalnimi parametri, razdeljen na posamezne gradnike, v odvisnosti od števila vozlišč.

Na sliki 5.7 so isti podatki za MPM2. Skupni čas se zmanjša, ker potrebujemo manj kvadraturnih točk na kvadraturno domeno. Pač pa je izračun vrednosti baznih funkcij v posamezni kvadraturni točki zahtevnejši in pomeni prevladujoč del skupnega časa. Hitra izvedba LU razcepa utežene momentne matrike P_W je zato bistvenega pomena.
Sklep: MPM2 je časovno manj zahtevna kot MPM1, pri čemer je delež časa za izračun vrednosti baznih funkcij večji.

Faktor $\log N$, ki nastopa v asimptotični časovni zahtevnosti, na grafih ni viden, marveč vse krivulje izgledajo kot linearna odvisnost. Slika 5.8 zato prikazuje čase, ki jih posamezni gradniki in celotna MPM2 porabijo za vsako vozlišče.



Slika 5.7: Čas, porabljen za MPM2 z optimalnimi parametri, razdeljen na posamezne gradnike, v odvisnosti od števila vozlišč.



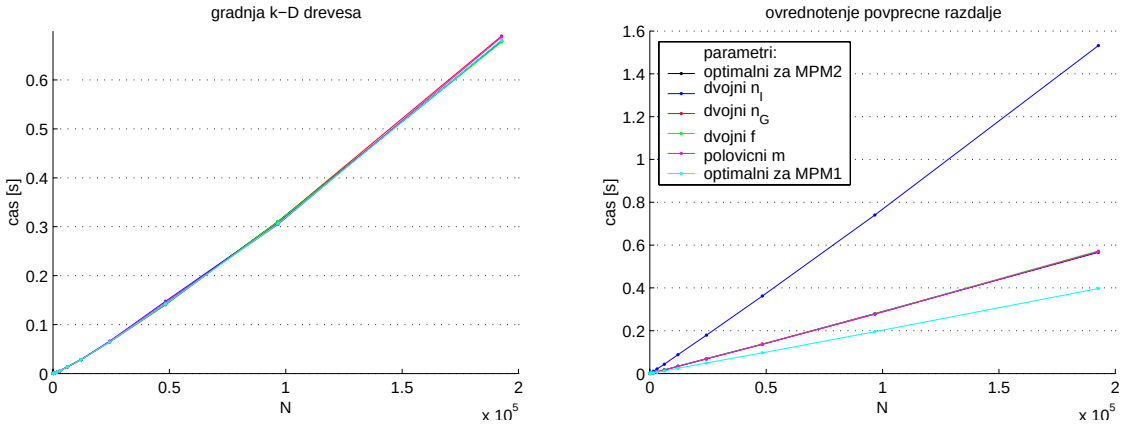
Slika 5.8: Časi, porabljeni za posamezne gradnike in celotno MPM2, deljeni s številom vozlišč.

Zahtevnost na vozlišče pri prvih dveh gradnikih je logaritemska. Tudi zahtevnost iskanja je od približno 400 vozlišč dalje logaritemska. Izmerjena zahtevnost na vozlišče za računanje baznih funkcij in sestavljanje matrik je v skladu s teorijo konstantna. V skupnem času se nelinearnost gradnje drevesa, ovrednotenja $\overline{d}$ in iskanja ne pozna, ker ti gradniki predstavljajo premajhen delež časa. Sodeč po asimptotičnih in izmerjenih časovnih zahtevnostih bi od približno 10 milijonov vozlišč dalje v skupnem času prevladovalo iskanje. Pri zadnjih treh gradnikih in 49 do 400 vozliščih izmerjena zahtevnost na vozlišče strmo raste, kar je najverjetneje posledica naraščajočega števila predpomnilniških zgrešitev.

5.4.2 Vpliv parametrov na zahtevnost gradnikov MPM

Oglejmo si za vsak gradnik, kako parametri MPM vplivajo na njegovo zahtevnost. Za referenco smo vzeli čase, dobljene s parametri, optimalnimi za MPM2. Čase smo izmerili še za spremenjene parametre:

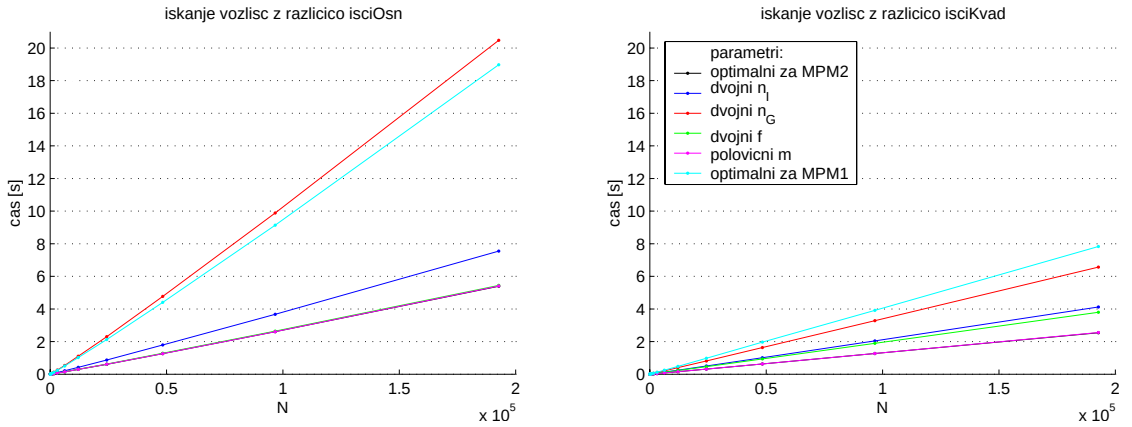
1. podvojen $n_I = 22$ namesto 11 tako, da povečamo α_S (4.1 namesto 2.6); ker mora f ostati enak, povečamo tudi α_Q (1.26 namesto 0.8),
2. podvojen $n_G = 6$ namesto 3,
3. podvojen $f = 4.14$ namesto 2.06 tako, da povečamo α_Q (1.9 namesto 0.8),
4. razpolovljen $m = 3$ namesto 6 tako, da zmanjšamo stopnjo aproksimacije MLS (1 namesto 2),
5. MPM1 z ustreznimi optimalnimi parametri.



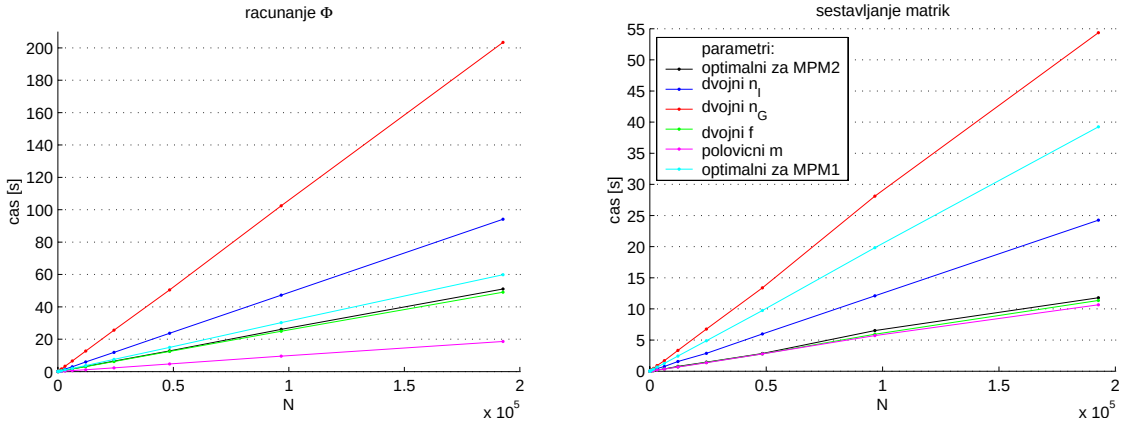
Slika 5.9: Vpliv parametrov na čas za gradnjo k-D drevesa (levo) in za ovrednotenje $\overline{d}$ na pravilni mreže (desno).

Slika 5.9 prikazuje odvisnost izmerjene časovne zahtevnosti gradnje k-D drevesa in ovrednotenja $\overline{d}$ od parametrov. Na gradnjo drevesa parametri sploh ne vplivajo. Tudi ovrednotenje povprečne razdalje se obnaša v skladu s teorijo, saj podvojitev n_I nekoliko več kot podvoji potrebni čas.

Čas, porabljen za iskanje vozlišč v nosilni domeni, za obe različici prikazuje slika 5.10. V različici išščiOsn podvojitev n_G početveri potrebni čas, podvojitev n_I pa ga poveča za manj kot dvakrat, ker je le eden od obeh členov v izrazu za asimptotično zahtevnost sorazmeren n_I. Različica išščiKvad je v vseh primerih hitrejša, odvisna pa je tudi od f, zato je pridobitev najmanjša v primeru podvojenega f in pri MPM1, kjer je f prav tako povečan.



Slika 5.10: Vpliv parametrov na čas za iskanje vozlišč v nosilnih domenah kvadraturnih točk z različico išščiOsn (levo) in z različico išščiKvad (desno).



Slika 5.11: Vpliv parametrov na čas za izračun baznih funkcij ter njihovih odvodov (levo) in sestavljanje matrik A in B (desno).

Na računanje vrednosti baznih funkcij (slika 5.11 levo) v skladu s teorijo kvadratno vpliva n_G in linearno n_I, medtem ko s polovičnim m računanje traja dobro tretjino namesto teoretično predvidene četrtine časa, kar lahko pripišemo manjši učinkovitosti LU razcepa in ostalih matričnih operacij na manjši velikosti matrik. Sestavljanje matrik (slika 5.11 desno) je približno kvadratno odvisno od n_G in linearno od n_I.

Sklep: Vplivi parametrov na časovno zahtevnost posameznih gradnikov se ujemajo z asimptotično zahtevnostjo.

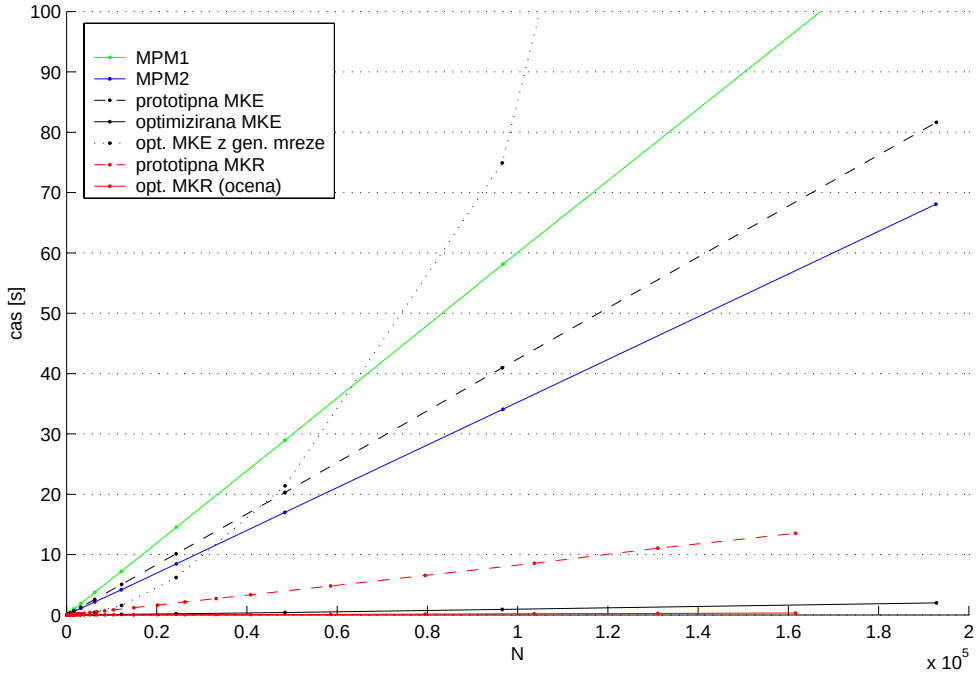
5.4.3 Primerjava s končnimi razlikami in končnimi elementi

Za prototipni izvedbi MKR in MKE v okolju Matlab smo izmerili čase, potrebne za generiranje sistemskih matrik na nekoliko drugačnem računalniku z enim procesorjem AMD athlon 64 3000+, operacijskim sistemom Windows XP in Matlabom verzije 6.1.0.450. Optimizirano izvedbo MKE smo izmerili na prej opisanih računalnikih s procesorji AMD opteron. MKR smo implementirali le prototipno in predpostavili, da bi z optimizirano izvedbo v jeziku C++ dosegli enako izboljšanje kot pri MKE. Zahtevnost morebitne optimizirane različice MKR smo torej ocenili tako, da smo pri vsaki vrednosti N čas prototipne izvedbe MKR delili z razmerjem časov prototipne in optimizirane izvedbe MKE.

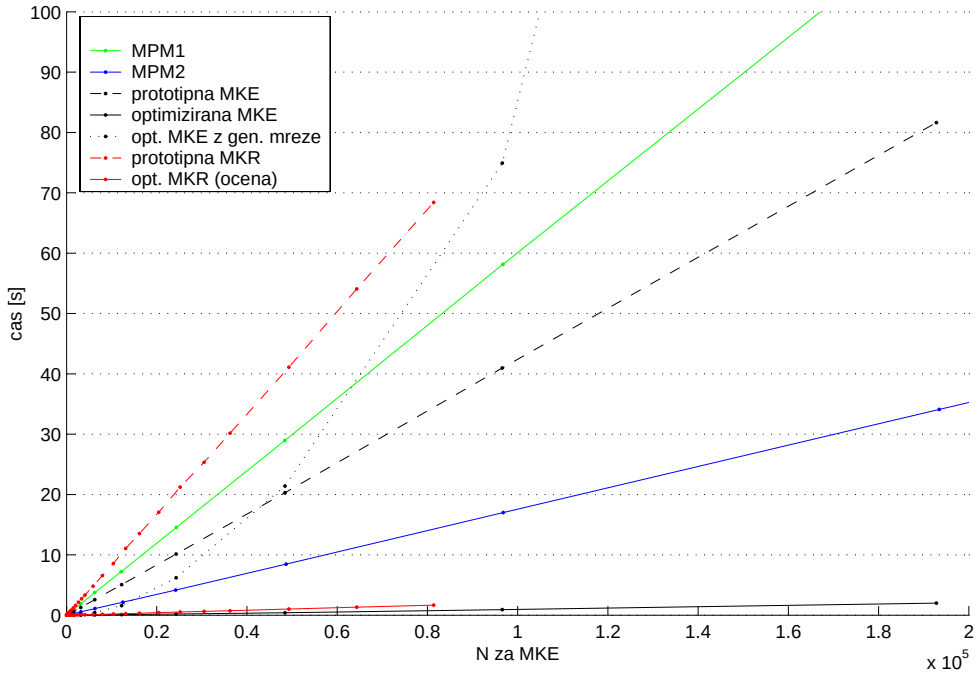
Za 24 000 in več vozlišč smo tudi izmerili čas generiranja mreže elementov s Femlabom na računalniku s procesorjem athlon 64. Iz meritev smo sklepali na asimptotično zahtevnost $O(N^2)$, zato smo jih aproksimirali s kvadratnim polinomom $T_{MREZA} = 10^{-8}N^2$ sekund in s pomočjo te ocene dobili približen čas generiranja manjših mrež, ki ga nismo mogli natančno izmeriti.

Primerjavo izmerjenih časov za sestavljanje linearnega sistema v odvisnosti od števila vozlišč prikazuje slika 5.12. Pri MKR je čas za sestavljanje sistema praktično zanemarljiv, zato je edini zahteven del reševanje sistema. Tudi pri MKE je čas majhen, vendar narašča hitreje kot linearno; na prikazanem intervalu vrednosti N je čas približno sorazmeren $N^{1.15}$. Za računsko najzahtevnejši del MKE, to je generiranje mreže elementov, bi bilo pri 10 000 in več vozliščih nujno uporabiti hitrejši algoritem od tega, ki je vgrajen v Femlab. Pri MPM je čas za generiranje vozlišč zanemarljiv, zahtevnost sestavljanja linearnega sistema pa je na prikazanem intervalu vrednosti N praktično linearna.

Za pošteno primerjavo metod je nujno upoštevati tudi njihovo natančnost. Kot smo ugotovili v razdelku 4.6.2, lahko čez palec ocenimo, da pri reševanju dvodimenzijske difuzijske enačbe MKR za isto natančnost potrebuje približno desetkrat več vozlišč kot MKE, MPM1 približno toliko kot MKE, MPM2 pa vsaj dvakrat manj. Slika 5.13 zato prikazuje normaliziran čas za sestavljanje sistema, ki je za MKR kar graf s slike 5.12, stisnjen po osi N za faktor 10, za MPM2 pa raztegnjen po osi N za faktor 2. Z MKR sploh ne moremo dobiti zelo natančne rešitve, ker potrebno število vozlišč in s tem prostorska zahtevnost postaneta prevelika.
Sklep: Zaključimo lahko, da uporaba MPM1 ni priporočljiva. MPM2 je hitrejša od MKE s Femlabovim algoritmom za generiranje mreže, če zahtevamo natančnost, za katero MKE potrebuje 10 000 ali več vozlišč. V našem testnem primeru z luknjo to pomeni absolutno napako, manjšo od 0.004. Z boljšim algoritmom za generiranje mreže bi bila MKE bistveno bolj konkurenčna glede hitrosti, vendar ima že zaradi same potrebe po mreži druge slabosti. MKR je najenostavnejša, vendar le navidez hitra, saj je zaradi potrebnega večjega števila vozlišč reševanje sistema dolgotrajnejše.



Slika 5.12: Izmerjen čas za sestavljanje linearnega sistema z MPM, MKE brez in z generiranjem mreže elementov ter MKR v odvisnosti od števila vozlišč.



Slika 5.13: Normaliziran čas, ki upošteva različno natančnost metod, za sestavljanje linearnega sistema z MPM, MKE brez in z generiranjem mreže elementov ter MKR.

Vložek človeškega dela

Še pomembnejši od golega računalniškega časa je potreben vložek človeškega dela, da metoda začne delovati. Ta je le pri MKR trivialen. Postopek priprave mreže končnih elementov mora nujno nadzorovati strokovnjak, ki navadno za to opravilo porabi večino časa [IOCP03, Owe98, Liu03]. Nasprotno generiranja vozlišč za MPM ni težko avtomatizirati. V ozadju MPM je bolj zapletena matematična formulacija kot pri MKR in MKE, vendar to vpliva le na težavnost prve izvedbe, ne pa na težavnost reševanja novih problemov. Človeško delo in znanje je potrebno predvsem za določitev parametrov, s katerimi bodo MPM dale dovolj natančen rezultat. Iskanje parametrov lahko delno avtomatiziramo, če poznamo dovolj natančno rešitev kakega problema, ki je do takšne mere podoben reševanemu problemu, da so optimalne vrednosti parametrov podobne.
Sklep: Tako MKE kot MPM zahtevata precejšen vložek človeškega dela. Pričakovati je, da se bo z nadaljnjim razvojem obeh metod potreba po človeškem delu zmanjšala, še posebno pri MPM.

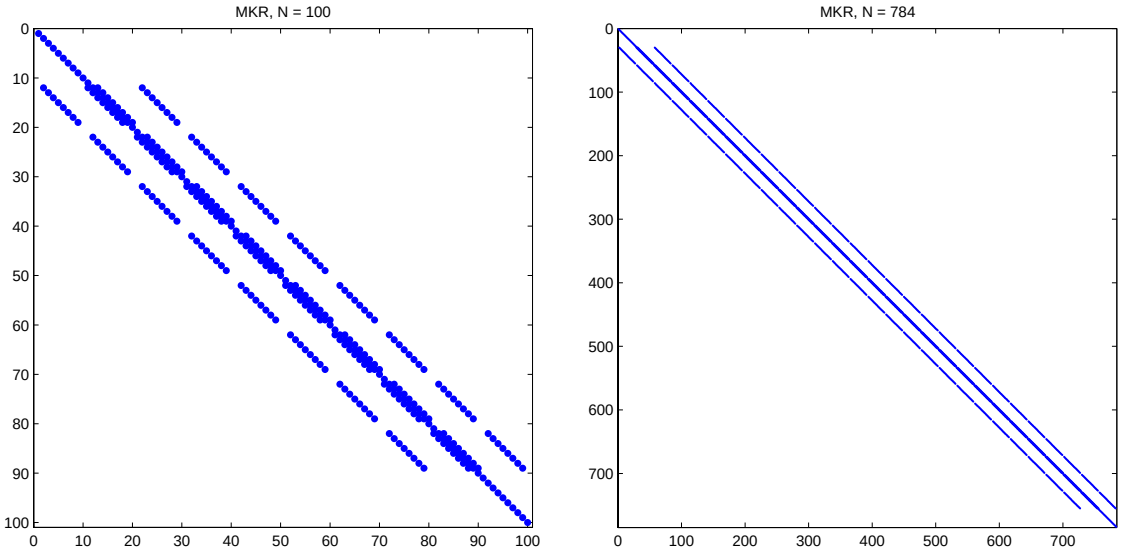
5.5 Zahtevnost reševanja linearnega sistema

Podrobna analiza zahtevnosti reševanja dobljenega linearnega sistema ni predmet tega dela, zato bomo le opisali nekatere lastnosti sistemske matrike A, ki omogočajo vpogled v zahtevnost reševanja. Morebitna simetričnost in pozitivna definitnost matrike dovoljujeta uporabo algoritmov, kot sta SOR in konjugirani gradienti. Število neničelnih elementov v matriki A in matriki predpogojevanja neposredno vpliva na čas, potreben za eno iteracijo konjugiranih gradientov, BiCGSTAB in podobnih algoritmov. Število potrebnih iteracij je odvisno od občutljivosti predpogojenega sistema [Hea02, GO93].

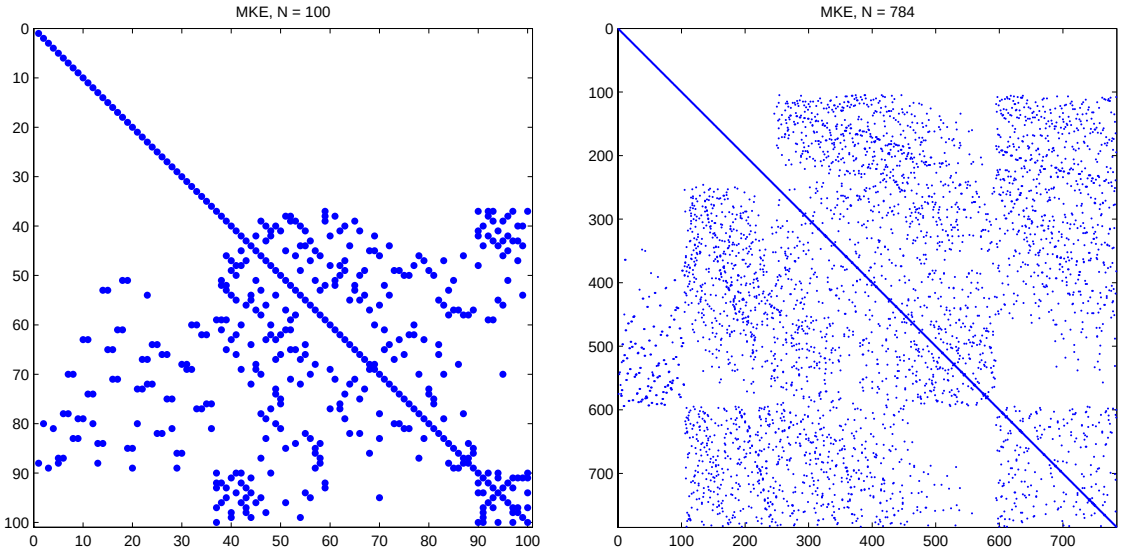
5.5.1 Metoda končnih razlik

Pri reševanju difuzijske enačbe z MKR na kvadratni domeni dobimo sistemsko matriko A, ki ima strukturo, prikazano na sliki 5.14. Število neničelnih elementov bomo označevali z $nnz(A)$ in je v primeru MKR približno $nnz(A_{MKR}(N)) \approx 5N$ (v resnici nekaj manj, ker enačbe potrebujemo le za notranja vozlišča). Na sliki so neničelni elementi prikazani z modrimi pikami in tvorijo diagonalni pas širine 3 ter dva pasova širine 1, oddaljena od diagonale za $\sqrt{N}$.

Iz vektorja neznank bi lahko odstranili robna vozlišča in njihov vpliv premestili na desno stran linearnega sistema, s čimer bi matrika postala simetrična in pozitivno definitna. V našem primeru to ni potrebno, ker za reševanje uporabljamo metodo BiCGSTAB.



Slika 5.14: Struktura neničelnih elementov sistemske matrike A za MKR s 100 vozlišči (levo) in s 784 vozlišči (desno).



Slika 5.15: Struktura neničelnih elementov sistemske matrike A za MKE s 100 vozlišči (levo) in s 784 vozlišči (desno).

5.5.2 Metoda končnih elementov

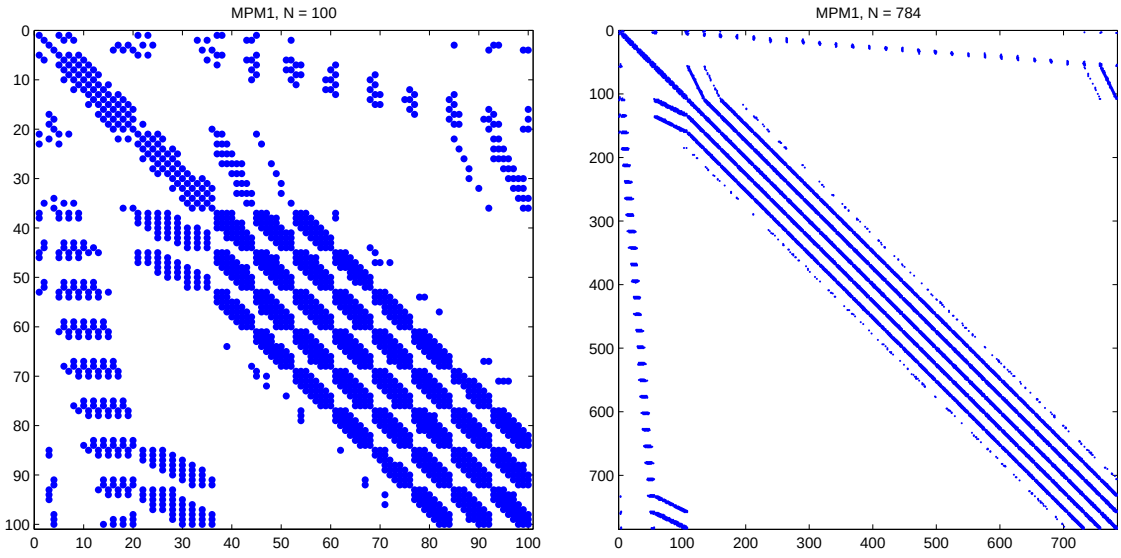
Z MKE pri reševanju istega problema dobimo sistemsko matriko A s približno $6.7N$ neničelnimi elementi, katerih položaj v matriki je odvisen od oštevilčenja vozlišč. Slika 5.15 prikazuje strukturo A pri oštevilčenju programa Femlab, ki robnim vozliščem dodeli nižje indekse kot notranjim.

Tudi pri MKE bi lahko na enak način kot pri MKR poskrbeli za simetričnost in pozitivno definitnost matrike. Odrezati bi morali vrstice, ki vsebujejo le diagonalni element, in pripadajoče stolpce. Tako bi obdržali le spodnji desni del matrik, za katerega na sliki vidimo, da ima simetrične položaje neničelnih elementov.

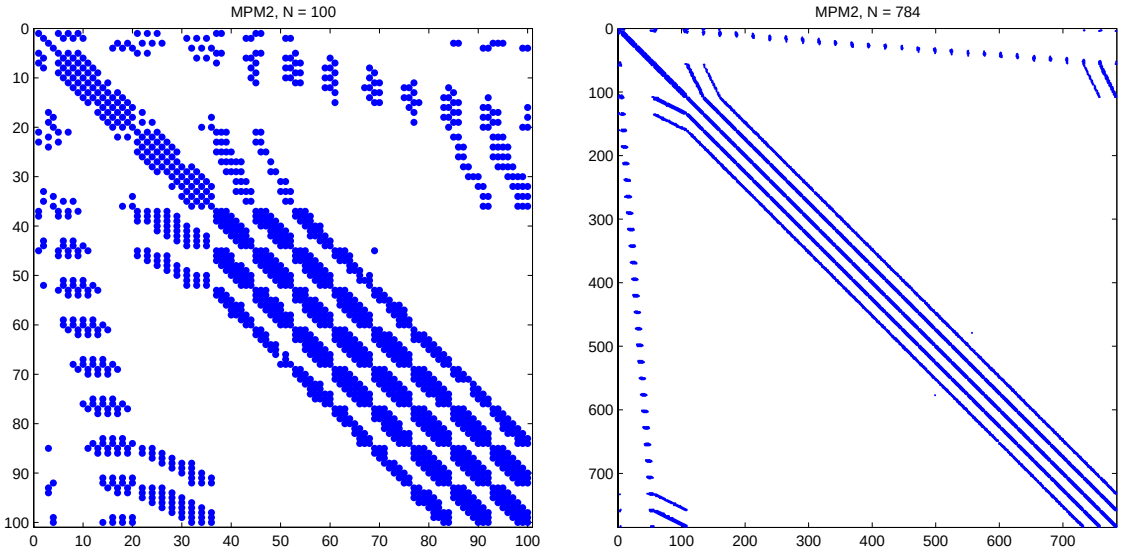
5.5.3 Mreže prosta metoda

Pri reševanju istega problema z MPM dobljena matrika A ima nekaj manj kot $f \cdot n_I \cdot N$ neničelnih elementov, in sicer $nnz(A_{MPM1}(N)) \approx 24N$ in $nnz(A_{MPM2}(N)) \approx 20N$. Strukturo matrike pri oštevilčenju vozlišč po x-koordinatah za MPM1 prikazuje slika 5.16 in za MPM2 slika 5.17.

Sistema ne moremo preformulirati tako, da bi dobili simetrično matriko, ker testne funkcije niso enake baznim. Poleg tega zaradi odsotnosti Kroneckerjeve δ lastnosti baznih funkcij in vsiljevanja robnih pogojev s kolokacijo ne moremo odstraniti robnih vozlišč iz vektorja neznank. Struktura matrik sicer na slikah izgleda skoraj simetrična, vendar se vrednosti elementov na obeh straneh glavne diagonale precej razlikujejo.



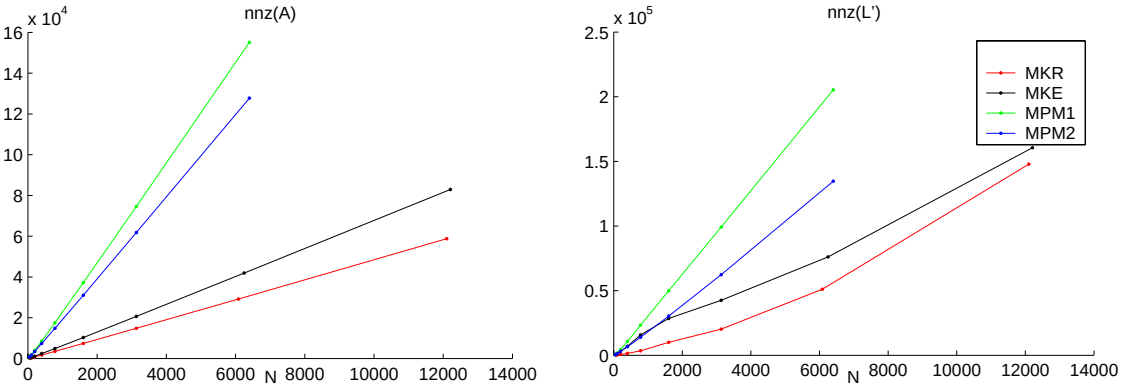
Slika 5.16: Struktura neničelnih elementov sistemske matrike A za MPM1 s 100 vozlišči (levo) in s 784 vozlišči (desno).



Slika 5.17: Struktura neničelnih elementov sistemske matrike A za MPM2 s 100 vozlišči (levo) in s 784 vozlišči (desno).

5.5.4 Primerjava lastnosti sistemskih matrik

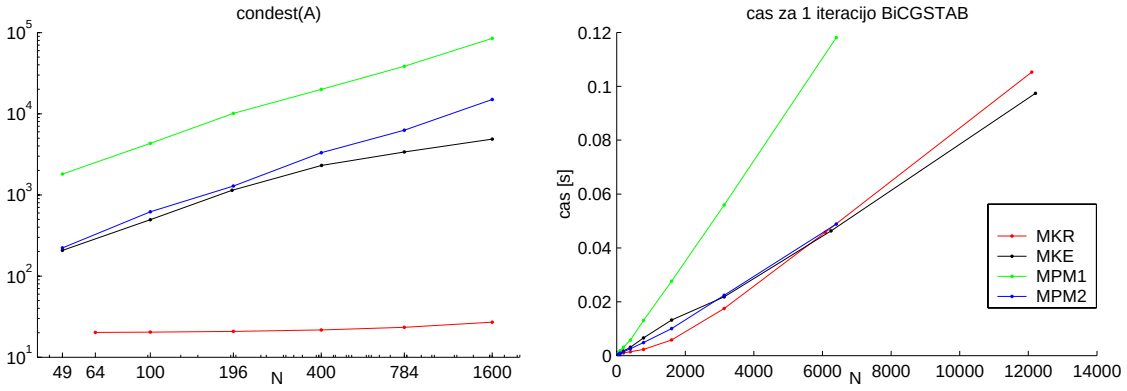
Na levem grafu na sliki 5.18 je primerjava števila neničelnih elementov $nnz(A)$ v sistemski matriki za vse tri metode.



Slika 5.18: Število neničelnih elementov $nnz(A)$ v matriki sistema (levo) in v faktorju nepopolnega razcepa $nnz(L')$ (desno) v odvisnosti od števila vozlišč za MKR, MKE in MPM.

Za predpogojevanje lahko uporabimo nepopoln LU razcep $L'U' \approx A$ s toleranco 10^{-6} ; število neničelnih elementov v faktorju L' je primerjano na desnem grafu na sliki 5.18. Pri MPM narašča s povečevanjem števila vozlišč linearno: $nnz(L'_{MPM1}(N)) \approx 32N$ in $nnz(L'_{MPM2}(N)) \approx 21N$. Pri MKR in MKE narašča hitreje kot linearno, a je vseeno opazno manjše kot pri MPM.

Število občutljivosti matrike A smo ocenjevali z Matlabovo funkcijo `condest`. Primerjavo med metodami kaže levi graf na sliki 5.19. Pri MKR lahko oceno števila občutljivosti linearno aproksimiramo kot $condest(A_{MKR}(N)) \approx N/220+20$. Pri MKE je ocena števila občutljivosti bistveno večja, vendar raste počasneje kot linearno. Pri MPM narašča nekoliko hitreje kot linearno, pri čemer je z MPM1 dobljeni sistem približno sedemkrat bolj občutljiv kot sistem, dobljen z MPM2.



Slika 5.19: Ocena $condest(A)$ števila občutljivosti (levo) in računski čas v sekundah, potreben za eno iteracijo metode BiCGSTAB s predpogojevanjem (desno) v odvisnosti od števila vozlišč za MKR, MKE in MPM.

Ocena števila občutljivosti za predpogojeni sistem $U'^{-1}L'^{-1}A$ je v testiranem intervalu približno $1 + 4 \cdot 10^{-5}N$ za MPM1 in $1 + 10^{-5}N$ za MPM2. Pri MKR je v testiranem intervalu manjša od 1.00001 in je na podlagi naših meritev niti ne moremo zvesto aproksimirati, vemo le, da z N narašča zelo počasi. Podobno je pri MKE ocena števila občutljivosti za predpogojeni sistem manjša od 1.0001. V vseh testiranih primerih je predpogojevanje dovolj uspešno, da metoda BiCGSTAB reši sistem že v eni iteraciji.

Desni graf na sliki 5.19 prikazuje čas za eno iteracijo metode BiCGSTAB na računalniku s procesorjem AMD athlon 64 3000+ in Matlabom verzije 6.1.0.450. Pričakovali smo, da bo trajanje matričnih operacij, uporabljenih v metodi BiCGSTAB (reševanje dveh trikotnih sistemov z matrikama L' in U' ter množenje vektorja z matriko A), odvisno le od števila neničelnih elementov in bo graf podoben obema grafoma na sliki 5.18. Podrobnejši poskusi so pokazali, da na primer reševanje trikotnega sistema z matriko L' pri MKR s 6084 vozlišči traja kar trikrat dlje kot pri MKE s 6247 vozlišči, čeprav ima matrika v drugem primeru 50% več neničelnih elementov kot v prvem.

Časa trajanja celotne iteracije torej ni moč teoretično predvideti. Pri MPM2 in MKE je skoraj enak, za MPM1 opazno večji, v teh treh primerih pa tudi približno linearno odvisen od N. Za sistem, dobljen z MKR, čas za eno iteracij narašča hitreje od linearnega in je pri velikem številu vozlišč celo daljši kot pri MKE, čeprav ima slednja v matrikah A, L' in U' več neničelnih elementov.

Sklep: Povzamemo lahko, da je časovna zahtevnost reševanja sistema z metodo Bi-

CGSTAB pri podobnem številu vozlišč za MKR, MKE in MPM2 v istem velikostnem razredu, za MPM1 pa nekajkrat večja kot za MPM2.

5.6 Razmerje med sestavljanjem in reševanjem linearnega sistema

Zavedati se moramo tudi razmerja potrebnega časa za sestavljanje sistemskih matrik in reševanje dobljenega sistema. Iz slike 5.12 na strani 108 ter desnega dela slike 5.19 lahko sklepamo, da je pri MKE časovna zahtevnost sestavljanja sistema v istem velikostnem razredu kot njegovo enkratno reševanje. Pri MPM1 je sestavljanje za dva, pri MPM2 pa za en velikostni razred bolj zahtevno od enkratnega reševanja. Navedena razmerja veljajo za reševanje časovno neodvisne PDE, kjer moramo sistem enkrat sestaviti in enkrat rešiti, in za časovno odvisno PDE na hitro spreminjajoči se domeni, kjer oboje ponavljamo v vsakem koraku.

V primeru reševanja časovno odvisne PDE na nespremenljivi domeni, kakršni so vsi naši primeri z difuzijsko enačbo, pa sistem le enkrat sestavimo in ga nato rešimo enkrat na časovni korak, zato se poveča pomembnost hitrosti reševanja. Pri dolgih simulacijah, sestavljenih iz deset tisoč in več časovnih korakov, predstavlja sestavljanje majhen ali celo zanemarljiv del celotnega časa.

Poglavje 6

Izvedba metod na vzporednih računalnikih

Najbolj naraven in enostaven način za izvedbo vzporednega računalniškega programa je uporaba enakega programa z različnimi podatki. Podatke, v našem primeru vozlišča, bomo enakomerno razdelili med vse razpoložljive procesorje, nato pa bo vsak od njih izračunal rešitev na svojem delu podatkov. Med izračunom bo občasno potrebna komunikacija - izmenjava vmesnih rezultatov.

V tem poglavju si najprej ogledamo, kako razdelimo podatke, da bosta tako gradnja sistema navadnih enačb kot njegovo reševanje učinkovito izrabili razpoložljive procesorje, in opišemo paralelizacijo metod končnih razlik in končnih elementov. Nato se posvetimo mreže prosti metodi, za katero predlagamo dva načina delitve domene in teoretično analiziramo njun vpliv na potrebno komunikacijo med procesorji pri gradnji sistema in med njegovim reševanjem. Predstavimo pohitritve, izmerjene na testnem skupku sedemnajstih dvoprocesorskih računalnikov. Izpeljemo eksplicitno formulo za napovedovanje pohitritve v odvisnosti od števila vozlišč in procesorjev ter njene napovedi primerjamo z meritvami. Na koncu analiziramo še zahtevnost komunikacije in njeno odvisnost od povezovalnega omrežja ter vpliv hitrosti procesorjev in komunikacije na vzporedno izvedbo programa.

6.1 Delitev domene

Kot smo povedali v prejšnjih poglavjih, reševanje PDE z MKR, MKE in MPM začnemo s sestavljanjem sistema navadnih enačb, ki ga nato rešimo. V vsakem časovnem koraku v primeru spremenljive domene ponovimo oboje, sicer pa le reševanje sistema. Princip paralelizacije z *delitvijo domene* je zategadelj pri vseh treh metodah podoben: vozlišča razdelimo na *poddomene* in vsako poddomeno s pripadajočimi vozlišči dodelimo enemu procesorju. Vsak procesor obdeluje vozlišča v svoji poddomeni in po potrebi z drugimi

procesorji izmenjuje podatke o ostalih vozliščih. Drugačen način paralelizacije, pri katerem bi vsak procesor opravil svoj del izračunov na celotnih podatkih, bi zahteval večjo količino komuniciranja in zato omejeno pohitritev [FJL+88]. Delitev domene je potrebno narediti na enem računalniku pred začetkom vzporednega dela postopka, zato je nizka računska zahtevnost algoritma za delitev pomembnejša od kakovosti delitve.

Sestavljanje sistemskih matrik A in B po delih je razmeroma preprosto. Vsak procesor sestavi tiste vrstice matrik, za katerih vozlišča je zadolžen. Edina komunikacija je začetna delitev. Pač pa moramo paziti, da so po končanem sestavljanju tako vozlišča kot sistemski matriki med procesorje razdeljeni tako, da lahko tudi reševanje sistema učinkovito paraleliziramo.

6.1.1 Paralelizacija reševanja sistema

Reševanje sistema z iterativno metodo zahteva v vsaki iteraciji izvajanje operacij, podobnih naslednjim [DHL03]:

1. množenje konstante s porazdeljenim vektorjem,
2. izračun vektorske norme,
3. množenje sistemske matrike in vektorja vozliščnih parametrov $\mathbf{u}$,
4. predpogojevanje, to je reševanje enostavnih redkih sistemov.

Prvi dve operaciji enostavno paraleliziramo ne glede na to, kako so elementi $\mathbf{u}$ porazdeljeni med procesorje, pri čemer za izračun norme potrebujemo *globalno komunikacijo* za seštevanje prispevkov vseh delov porazdeljenega vektorja [GO93].

Vzporedno množenje matrike z $\mathbf{u}$ pojasnimo s primerom: naj bo matrika

$$A = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & -2 \end{bmatrix}, \qquad \text{torej} \qquad A\mathbf{u} = \begin{bmatrix} u_1 - u_3 \\ u_2 \\ u_1 - 2u_3 \end{bmatrix}.$$

Če je procesor p_1 zadolžen za neznanko u_1, potrebuje poleg nje še prvo vrstico matrike in vrednost neznanke u_3. Slednjo bo zanjo zadolženi procesor spremenil, zato bo po končanem množenju potrebna komunikacija. V splošnem mora imeti procesor, ki je zadolžen za vozliščni parameter u_i vozlišča $\mathbf{x}_i$, dostop do celotne i-te vrstice matrike in do vozliščnih parametrov u_j vseh *sosedov* vozlišča $\mathbf{x}_i$, pri čemer sta vozlišči $\mathbf{x}_i$ in $\mathbf{x}_j$ sosednji, če velja $A_{i,j} \neq 0$. Vsak procesor mora vsakič sprejeti novo vrednost sosednjih vozliščnih parametrov od njihovih lastnikov [GO93].

Uporaba predpogojevanja zahteva izračun vrednosti neznank po zaporedju njihovih odvisnosti. V primeru spodnje trikotnega sistema najprej izračunamo prvo neznanko, s pomočjo njene vrednosti drugo in tako dalje do zadnje, ki je odvisna od vseh ostalih. Izbrati moramo tak način predpogojevanja, ki se ga da učinkovito paralelizirati – graf

odvisnosti ne sme imeti povezav med neznankami, ki pripadajo različnim procesorjem. Nepopolni *LU* razcep ni primeren, so pa znani drugi pristopi, ki ustrezajo navedenim zahtevam, poleg tega pa omogočajo tudi vzporedni izračun matrike za predpogojevanje [Haa98].

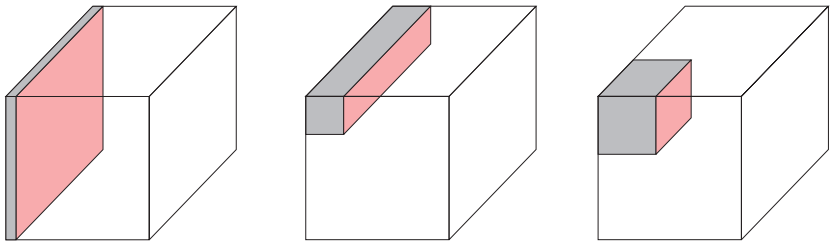
Vozlišča morajo biti med procesorje razdeljena tako, da množenje matrike z vektorjem zahteva čimmanj komunikacije, zato morajo meje med poddomenami sekati čimmanj povezav v grafu sosednosti vozlišč. Poleg tega mora biti število vozlišč v vseh procesorjih podobno, da bo delo porazdeljeno enakomerno [FJL+88, HE88]. S takšno razdelitvijo bo tudi predpogojevanje učinkovito [Haa98].

Pri reševanju velikega sistema na vzporednem računalniku zmerne velikosti, recimo z nekaj deset procesorji in nekaj tisoč neznankami na procesor, bomo z upoštevanjem gornjih priporočil dosegli vzporedno učinkovitost, večjo od 0.7 [ŠT03, Haa98].

6.1.2 Delitev domene pri metodi končnih razlik

Sosednost vozlišč je pri MKR kar geometrijska sosednost. V primeru reševanja dvodimenzijske difuzijske enačbe s Crank-Nicolsonovo shemo ima vsako vozlišče štiri sosede (šest v treh dimenzijah), v bolj zapletenih primerih pa lahko tudi več [Hea02], recimo 8 (26 v treh dimenzijah). Vedno pa je najugodneje razdeliti domeno na povezane poddomene s čim manjšo površino robov. Najenostavneje je, če so vsi robovi poddomen vzporedni z osmi oziroma ravninami koordinatnega sistema [FJL+88].

Najbolj naravne možnosti so *enodimenzionalna* (na rezine), *dvo-* ali *tridimenzionalna* (na kocke) delitev [GGKK03], kot prikazuje slika 6.1. Poddomena enega izmed procesorjev je obarvana sivo in ima v vseh treh primerih enako prostornino. Dvo- in tridimenzionalna razdelitev sta smiselni le, če se da število procesorjev razbiti na dva oziroma tri faktorje podobne velikosti, torej zahtevata vsaj štiri oziroma osem procesorjev.

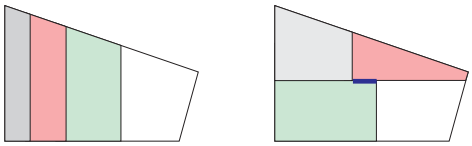


Slika 6.1: Eno-, dvo- in tridimenzionalna razdelitev domene.

Ob predpostavki, da vsak procesor hkrati komunicira z vsemi sosedi s polno hitrostjo, je skupni čas komunikacije sorazmeren površini največje ploskve poddomene (rožnata barva na sliki). V primeru d-dimenzionalne delitve domene velikosti $N = k \times k \times k$ vozlišč med p procesorjev je površina enaka $k^2 p^{(1-d)/d}$. Teoretično je tako boljša večdimenzionalna

razdelitev. Z večanjem števila procesorjev se razlika med načini razdelitve povečuje, z večanjem števila vozlišč pa zmanjšuje, ker čas računanja prevlada nad časom komunikacije [FJL+88]. Če hkratna komunikacija z več sosedi ni možna, je skupni čas komunikacije sorazmeren vsoti površin vseh ploskev poddomene.

Pomembna prednost enodimenzionalne delitve je enostavnost prilagajanja velikosti poddomen. Na domeni nepravilne oblike so lahko poddomene različno široke, tako da je volumen poddomen podoben in so izračuni na vseh poddomenah približno enako zahtevni (slika 6.2 levo). Dvo- in tridimenzionalna delitev se v tem primeru zapleteta, ker bodisi robovi poddomen postanejo poševni ali pa se njihovi vogali zamaknejo, zaradi česar se poveča število sosedov posameznih poddomen [FJL+88]. Na desnem delu slike 6.2 je z modro barvo prikazana meja med poddomenama, ki pri enakomerni razdelitvi nista bili sosednji. Vsak procesor si mora podatke izmenjevati z lastniki sosednjih poddomen. Če so računalniki povezani v kvadratno mrežo, poteka komunikacija med lastnikoma zelene in rožnate domene posredno prek enega od ostalih procesorjev namesto neposredno, zato je nekoliko počasnejša [GGKK03].



Slika 6.2: Prilagajanje velikosti poddomen pri enodimenzionalni (levo) in dvodimenzionalni delitvi (desno).

6.1.3 Delitev domene pri metodi končnih elementov

Pri MKE sta vozlišči sosednji, če sta del istega elementa [Kat03]. Postopek generiranja mreže zagotavlja neizrojenost elementov, zato velja podobno kot pri MKR: če so poddomene povezane in imajo čim manjšo površino robov, bodo njihovi robovi sekali malo povezav med vozlišči. Zaradi dodatne zahteve, da robovi poddomen ne smejo sekati morebitnih področij z zgoščenimi vozlišči, se za delitev običajno uporabljajo kar splošni aproksimacijski algoritmi za *razdeljevanje grafov* (angl. graph partitioning), kot je [KK98, KŠR02]. Uporaba aproksimacijskih algoritmov je nujna, ker je problem *NP-poln* [KŠR02].

6.2 Vzporedna mreže prosta metoda

Iz optimizirane zaporedne izvedbe MPM v jeziku C++ smo razvili več vzporednih različic. Za komunikacijo smo uporabili standardni vmesnik za komunikacijo z izmenjavo sporočil MPI. Vse različice vzporednih programov najprej razdelijo vozlišča med procesorje, nato

pa na vsaki poddomeni uporabijo algoritem SESTAVISISTEMMPM, opisan na strani 100. Rezultat vzporednih različic je v vsakem procesorju k-D drevo z njegovimi in potrebnimi sosednjimi vozlišči ter pripadajoče vrstice matrik A in B.

6.2.1 Delitev domene

Pri MPM relacija sosednosti med vozlišči ni simetrična [Liu03], zato raje govorimo o *odvisnosti*. Vozlišče $\mathbf{x}_i$ je odvisno od vozlišča $\mathbf{x}_j$, če je $A_{i,j} \neq 0$; to pomeni, da je bodisi vozlišče $\mathbf{x}_i$ robno in $\mathbf{x}_j \in \Omega_S(\mathbf{x}_i)$ ali $\mathbf{x}_i$ ni robno in $\mathbf{x}_j$ leži v nosilni domeni vsaj ene izmed kvadraturnih točk v $\Omega_I(\mathbf{x}_i)$.

Spet velja, da morajo robovi poddomen sekati čimmanj povezav v grafu odvisnosti. Podobno kot pri MKE to pomeni, naj bodo poddomene povezane in imajo čim manjšo površino robov. Matrika A je pri MPM bolj polna kot pri MKE, zato ima graf odvisnosti več povezav in so lahko aproksimacijski algoritmi za razdelitev grafa preveč časovno potratni. Vseeno se takšen, za MKE prirejen način delitve domene včasih uporablja [DUAL00]. Po drugi plati so vozlišča MPM običajno dobljena z enostavnim postopkom, zato lahko izluščimo informacijo o področjih z zgoščenimi vozlišči. Poddomene nato definiramo tako, da se njihove meje izognejo takšnim področjem.

V primeru difuzijske enačbe podrobnosti v obliki domene ne vplivajo bistveno na rešitev, zato so področja z zgoščenimi vozlišči smiselna le ob robovih. V našem primeru smo vozlišča zgostili le v bližini luknje, pa še tam ne tako močno, da bi morali to upoštevati pri določanju poddomen. Za primerjavo smo preizkusili dva načina delitve domene.

Enodimenzionalna delitev

Seznam vseh vozlišč uredimo po y-koordinati in razrežemo na p enako dolgih delov (če N ni deljivo s p, se dolžine nekaterih delov razlikujejo za eno vozlišče). Delitev po y izberemo zato, da so pravokotniki, očrtani poddrevesom v k-D drevesu, bližje kvadratnim, saj k-D drevo vozlišča najprej deli po x-koordinati.

Hierarhična delitev

Uravnoteženost k-D drevesa lahko izkoristimo za dvodimenzijsko delitev na domene, ki vsebujejo skoraj enako število vozlišč. *Hierarhična delitev* na p poddomen je delitev na tista poddrevesa k-D drevesa, ki imajo svoje korene na globini $\log p$ (spomnimo se, da globine štejemo od 0 naprej). Poleg tega moramo vsako vozlišče, ki je v drevesu višje od $\log p$, dodeliti v natanko eno izmed poddomen.

Vhodni podatki rekurzivnega algoritma za delitev na poddomene so tako: drevesno vozlišče v, število procesorjev p (ki mora biti potenca števila 2), identifikacijska številka

procId procesorja (procesorje štejemo od 0 do $p-1$), katerega poddomeno iščemo, in seznam S vseh vozlišč, ki so v drevesu višje od globine vozlišča v. Rezultat algoritma je pravokotna poddomena Ω_P. Pokličemo ga tako, da podamo koren predhodno zgrajenega k-D drevesa za v in prazen seznam za S.

```
ALGORITEM najdiHierarhPoddomeno(v, p, procId, S)
 1  if p = 1
 2    then Ω_P ← v.podrocje ∪ S
 3         // pravokotnik, očrtan vsem vozliščem iz poddrevesa v in seznama S
 4    else //če je procId sod, naj bo Ω_P podmnožica unije seznama S in levega poddrevesa
            vozlišča v, sicer pa podmnožica unije seznama S, vozlišča v in njegovega desnega
            poddrevesa;
 5         //še prej iz seznama S izbriši vsa vozlišča, ki glede na delitev v drevesnem vozlišču
            v ne spadajo v tisto poddrevo, ki ga boš vrnil
 6         for each x_i ∈ S
 7         do if v.globina mod 2 = 0 ∧
 8                ((procId mod 2 = 0 ∧ x_i ≥ v.x) ∨ (procId mod 2 = 1 ∧ x_i < v.x))
 9                then S.IZBRISI(x_i)
10              if v.globina mod 2 = 1 ∧
11                ((procId mod 2 = 0 ∧ y_i ≥ v.y) ∨ (procId mod 2 = 1 ∧ y_i < v.y))
12                then S.IZBRISI(x_i)
13         if procId mod 2 = 0
14           then Ω_P ← najdiHierarhPoddomeno(v.levi, p/2, ⌊procId/2⌋, S)
15           else Ω_P ← najdiHierarhPoddomeno(v.desni, p/2, ⌊procId/2⌋, S ∪ v)
```

Ker se p vsakič prepolovi, se skupaj izvede $\log p$ rekuzivnih klicev, preden je izpolnjen pogoj v vrstici 1. Zanka v vrsticah 6-12 in izračun unije v vrstici 2 porabita največ $O(|S|)$, vse ostale vrstice pa $O(1)$ časa. Ob vsakem klicu smo v seznam $|S|$ dodali največ eno vozlišče, zato vedno velja $|S| \leq \log p$ in je asimptotična časovna zahtevnost iskanja poddomene enega procesorja enaka $O(\log^2 p)$, zahtevnost iskanja poddomen vseh procesorjev pa $O(p \log^2 p)$. Če naj bo vzporedno izvajanje smiselno, mora veljati $p << N$, zato je čas za iskanje poddomen zanemarljiv v primerjavi s časom gradnje drevesa.

6.2.2 Relacija odvisnosti

Pri delitvi vozlišč med procesorje mora vsak poleg svojih vozlišč dobiti tudi vsa ostala, od katerih so njegova odvisna. Vendar relacija odvisnosti ni ne trivialna kot v MKR niti podana z mrežo kot v MKE, marveč sledi iz položajev vozlišč in velikosti kvadraturnih in nosilnih domen in torej v času delitve vozlišč ni znana [Liu03]. Preostane nam le oceniti takšno zgornjo mejo d_{max} največje povprečne razdalje med vozlišči, da je vsako vozlišče iz poddomene Ω_P odvisno le od tistih, ki so bodisi znotraj Ω_P ali pa od nje oddaljena za največ $(\alpha_S + \alpha_Q) \cdot d_{max}/2$. Med začetnim razdeljevanjem podatkov vsak procesor poleg vozlišč njegove poddomene prejme še ta dodatna vozlišča.

Ocena d_{max}

V primeru enodimenzionalne delitve moramo d_{max} oceniti glede na postopek generiranja vozlišč. S postopkom, predstavljenim v razdelku 4.3.1, dobimo vozlišča, ki so v notranjosti domene med seboj oddaljena približno za d. Po definiciji povprečne razdalje iz enačbe 3.14 na strani 49 je $\overline{d}$ blizu roba enaka $d\sqrt{2}$, zato lahko z dovolj rezerve predpostavimo: $d_{max} = 2d$. Odvisno od postopka generiranja vozlišč bi bil lahko d_{max} na različnih delih domene tudi različen.

Če bi vozlišča generirali z bolj naključnim postopkom, bi bilo potrebno vzeti več rezerve. V skrajnem primeru bi lahko preprosto določili $d_{max} = \infty$, tako da bi vsi procesorji poleg vozlišč v svoji poddomeni dobili tudi vsa druga vozlišča.

Hierarhična delitev domene zahteva, da najprej zgradimo k-D drevo z vsemi vozlišči. V tem primeru si lahko privoščimo tudi ovrednotenje $\overline{D}$ na pravilni mreži. Ker se za izračun $\overline{d}$ uporablja interpolacija vrednosti v $\overline{D}$, zagotovo velja $d_{max} \leq \max_{i,j} \overline{D}_{i,j}$.

Komunikacija med reševanjem sistema

Morebitna slaba ocena d_{max} je sprejemljiva, ker je za začetno razpošiljanje podatkov o vozliščih potrebno razmeroma malo komuniciranja v primerjavi s količino računanja med sestavljanjem linearnega sistema. Pač pa je komunikacija pogosto omejujoč faktor med vzporednim reševanjem sistema, zato si ne moremo privoščiti nepotrebnega izmenjevanja vozlišč. Že med sestavljanjem linearnega sistema zato vsak procesor zgradi tudi seznam vozlišč, ki jih potrebuje za svoje izračune. Če pozna še poddomene ostalih procesorjev, lahko ob vsaki izmenjavi podatkov zahteva samo potrebna vozlišča od njihovih lastnikov.

6.3 Pohitritev mreže proste metode

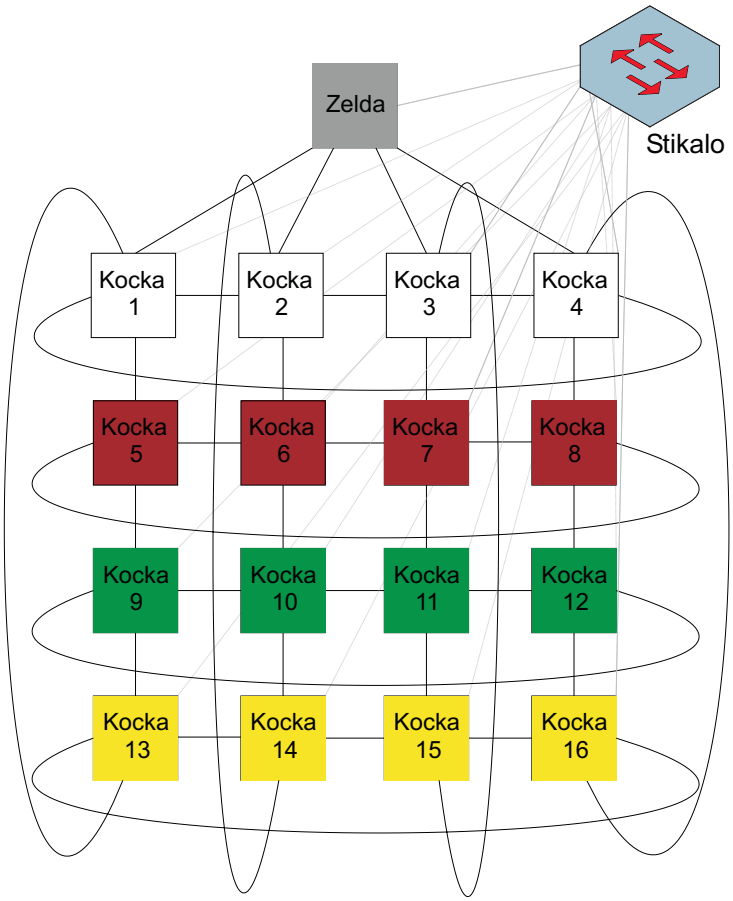
6.3.1 Testni skupek

Vse tri vzporedne različice programa za gradnjo linearnega sistema z MPM (različice s hierarhično in enodimenzionalno delitvijo domene, pri slednji še z izbirama $d_{max} = 2d$ ter $d_{max} = \infty$) smo preizkusili na testnem vzporednem računalniku – skupku, sestavljenem iz 17 računalnikov z naslednjo konfiguracijo:

- dva procesorja AMD opteron 244 s frekvenco 1.8 GHz, 128 KB prvonivojskega podatkovnega predpomnilnika in 1 MB drugonivojskega skupnega predpomnilnika,
- 1024 MB delovnega pomnilnika,
- šest mrežnih vmesnikov tipa Ethernet s hitrostjo 1 Gb/s,
- operacijski sistem Fedora Core 2 linux z jedrom 2.6.8-1.521smp,

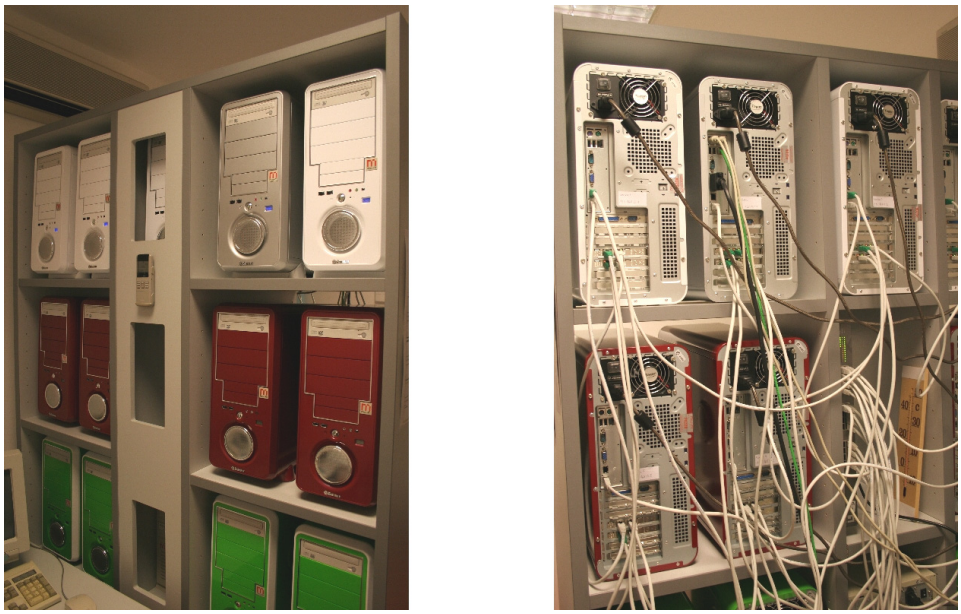
- prevajalnik `gcc` verzije 3.3.3,
- knjižnica LAM/MPI verzije 7.0.3.

Povezave med računalniki so shematično prikazane na sliki 6.3, na sliki 6.4 pa sta fotografiji sprednje in zadnje strani skupka. Barve na shemi ustrezajo dejanskim barvam ohišij računalnikov.



Slika 6.3: Neposredne povezave med računalniki v testnem skupku.

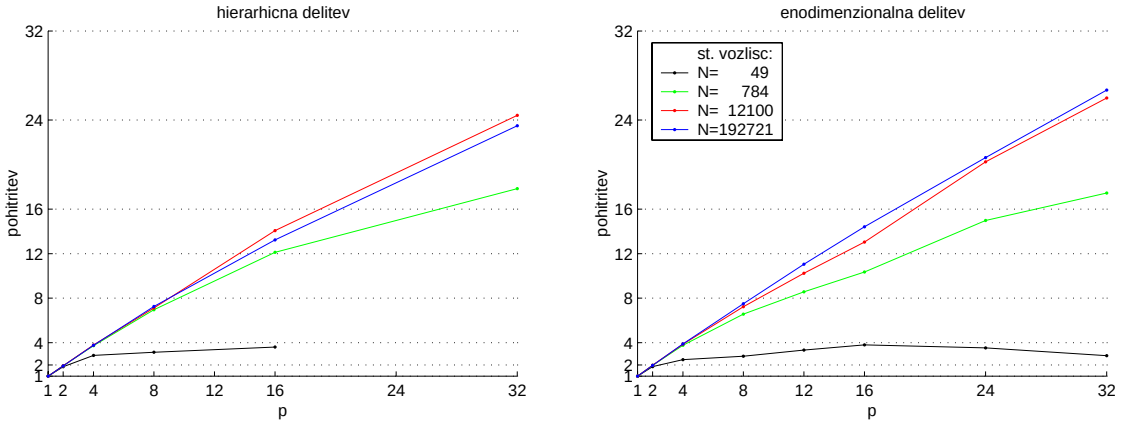
16 računalnikov "kocka1" do "kocka16" je povezanih v dvodimenzionalno toroidno mrežo, sedemnajsti "zelda" pa neposredno s prvimi štirimi. Vsi računalniki so povezani še s 24-vratnim omrežnim stikalom hitrosti 1 Gb/s. Usmerjevalne tabele so napisane tako, da se za komunikacijo med računalniki uporabljajo bodisi neposredne povezave, za en *skok* (angl. hop), ali dve zaporedni neposredni povezavi preko vmesnega računalnika za komuniciranje z računalnikom, oddaljenim za dva skoka. Komunikacija med ostalimi pari, oddaljenimi za več kot dva skoka, poteka preko omrežnega stikala [RŠT04, RTŠ05a].



Slika 6.4: Zgornji del skupka s sprednje (levo) in zadnje strani (desno).

6.3.2 Izmerjene pohitritve

Glede na rezultate predhodno opravljene analize, ki kažejo prednosti MPM2, smo se omejili le na to različico in uporabili zanjo optimalne parametre. Slika 6.5 prikazuje izmerjene pohitritve na testnem skupku v odvisnosti od števila uporabljenih procesorjev za različno število vozlišč in za oba načina delitve domene. Enodimenzionalna delitev z $d_{max} = 2d$ je vseskozi rahlo boljša, razen na 16 procesorjih s srednjim številom vozlišč. Hierarhična delitev zaostaja predvsem zaradi časa, potrebnega za gradnjo k-D drevesa na procesorju, kjer generiramo vozlišča. V tem času vsi ostali procesorji čakajo na svoja vozlišča. Iz



Slika 6.5: Izmerjene pohitritve na testnem skupku za oba načina delitve domene.

istega razloga doseže program s hierarhično delitvijo manjšo pohitritev pri večjem številu vozlišč, saj smo v prejšnjem poglavju ugotovili, da čas gradnje drevesa raste z N nekoliko hitreje kot celotni čas izvajanja. S poskusi smo potrdili, da je enodimenzionalna delitev z oceno $d_{max} = \infty$ vsaj za 20 % počasnejša od obeh ostalih različic. Če ne moremo s preprostim algoritmom dobiti dobre ocene d_{max}, je torej bolje uporabiti hierarhično delitev domene.

Čas, ki ga porabi vzporedni program za sestavljanje linearnega sistema z MPM2, je sestavljen iz treh delov:

1. priprave na delitev domene, ki se izvaja zaporedno na računalniku, ki je generiral vozlišča,
2. razdeljevanja vozlišč med procesorje, ki je omejeno s hitrostjo komunikacije,
3. vzporednega dela, kjer vsak procesor neodvisno obdeluje svoja vozlišča.

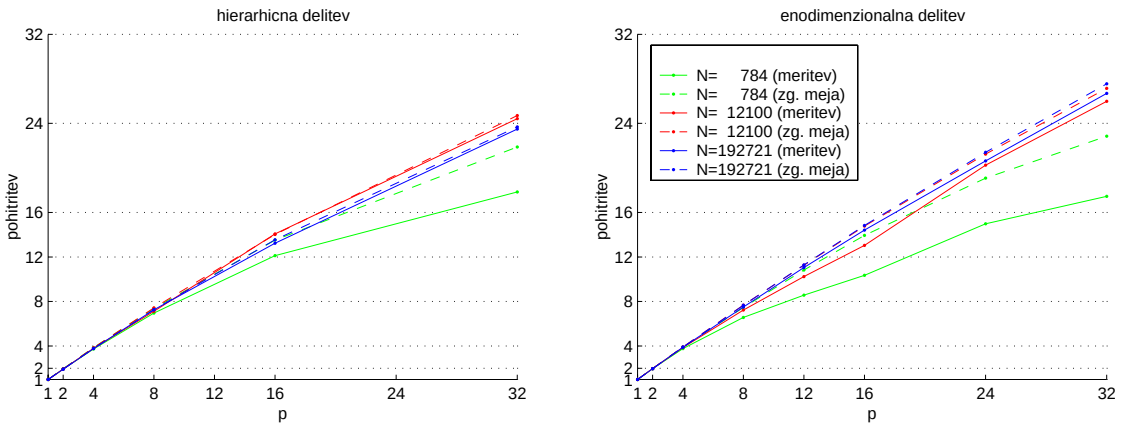
Prva dva dela omejujeta pohitritev po *Amdahlovem zakonu* [Kod00]. Tretji del traja, dokler dela ne opravi zadnji izmed procesorjev. Čas za tretji del je teoretično obratno sorazmeren številu procesorjev, v resnici pa nekaj večji zaradi ne povsem enakomerne razdelitve podatkov (razen, če je N deljiv s p) ter prekinitev in ostalih opravil, ki jemljejo procesorski čas.

Označimo s $t(N, p)$ celotni čas izvajanja vzporednega programa v odvisnosti od števila vozlišč N in števila procesorjev p. S $t_{zap}(N, p)$ označimo vsoto časov prvih dveh delov programa, torej $t(N, p) = t_{zap}(N, p) +$čas tretjega, vzporednega dela. S $t(N, 1)$ označimo čas izvajanja zaporedne izvedbe programa, s $t_{zap}(N, 1)$ pa tisti del zaporedne izvedbe, ki je v vzporedni izvedbi vključen v $t_{zap}(N, p)$.

Za vsak N in p smo izmerili $t_{zap}(N, p)$ in iz njega po Amdahlovem zakonu izračunali spodnjo mejo t_{min} časa za celotni program ter zgornjo mejo S_{max} pohitritve:

$$t_{min}(N, p) = \frac{t(N, 1) - t_{zap}(N, 1)}{p} + t_{zap}(N, p), \qquad S_{max}(N, p) = \frac{t(N, 1)}{t_{min}(N, p)}. \tag{6.1}$$

Tako izračunane zgornje meje so prikazane kot črtkane črte na sliki 6.6, za primerjavo pa so poleg tudi izmerjene pohitritve s slike 6.5. S hierarhično delitvijo se zgornji meji zelo približamo, razen pri 784 vozliščih, kar je za vzporedno računanje sorazmerno majhno število. Pri enodimenzionalni delitvi so zgornje meje višje, ker zaporedni del programa ne vključuje gradnje k-D drevesa, vendar se na račun manjše pohitritve vzporednega dela programa zgornji meji ne približamo tako kot pri hierarhični delitvi.
Sklep: Pohitritve so po pričakovanjih nekoliko nižje od linearnih. Pri 784 vozliščih je vzporedna učinkovitost pri uporabi vseh procesorjev testnega skupka okrog 0.55. Pri več kot 10 000 vozliščih je s hierarhično delitvijo učinkovitost okrog 0.75, z enodimenzionalno pa okrog 0.82. Enodimenzionalna delitev z $d_{max} = \infty$ deluje slabo.



Slika 6.6: Primerjava zgornje meje pohitritve z izmerjenimi vrednostmi.

6.3.3 Modeliranje pohitritve

Poskusimo z aproksimacijo izmerjenih časov dobiti eksplicitno formulo – model pohitritev MPM v odvisnosti od N in p. Omejimo se na hierarhično delitev domene, ki se obnaša bolj po pričakovanjih, je splošneje uporabna za nepravilne oblike domene in spremenljivo gostoto vozlišč, hkrati pa je ugodnejša za reševanje dobljenega sistema. Dobljena aproksimacija verjetno ne bo zelo natančna, vseeno pa bo omogočala čez palec vnaprej oceniti obnašanje vzporednega programa.

Čas izvajanja zaporednega programa

S slike 5.8 na strani 104 je razvidno, da enostavna ocena časa izvajanja zaporednega programa $\hat{t}(N,1) = 3.7 \cdot 10^{-4} N$ s na intervalu od 400 do 190 000 vozlišč zgreši izmerjeni čas za največ nekaj odstotkov. Glede na asimptotične in izmerjene časovne zahtevnosti posameznih gradnikov lahko predpostavimo, da ocena velja vsaj do enega milijona vozlišč.

Čas za pripravo delitve in delitev vozlišč

Na levem delu slike 6.7 je prikazana odvisnost t_{zap} od p. Zaradi preglednosti je y-os rdeče krivulje prikazana 10-krat povečano, zelene pa 50-krat. Za $N = 192\,721$ lahko uporabimo model $\hat{t}_{zap}(192\,721, p) = 0.76 + 1.83/p$, za $N = 12\,100$ pa $\hat{t}_{zap}(12\,100, p) = \hat{t}_{zap}(192\,721, p)/22$. Oba modela sta na sliki prikazana s črtkanimi črtami.

Za majhne p velja razmerje 1:22 tudi med $t_{zap}(12\,100, p)$ in $t_{zap}(784, p)$, za $p \geq 8$ pa model ne velja več. Razmerje časov 1:22 pri razmerju števila vozlišč 1:15 pomeni odvisnost od števila vozlišč $\hat{t}_{zap}(N) = K \cdot N^{\log 22/\log 15} = K \cdot N^{1.15}$. Z združitvijo odvisnosti od p in N

dobimo model:

$$\hat{t}_{zap}(N,p) = \left(0.76 + \frac{1.83}{p}\right) \cdot \left(\frac{N}{192\,721}\right)^{1.15},$$

ki zagotovo velja vsaj na intervalu $12\,100 < N < 192\,000$, $2 \leq p \leq 32$. Čas $t_{zap}(N,1)$ je bil v vseh meritvah s hierarhično delitvijo domene približno enak $0.4\,t_{zap}(N,2)$.

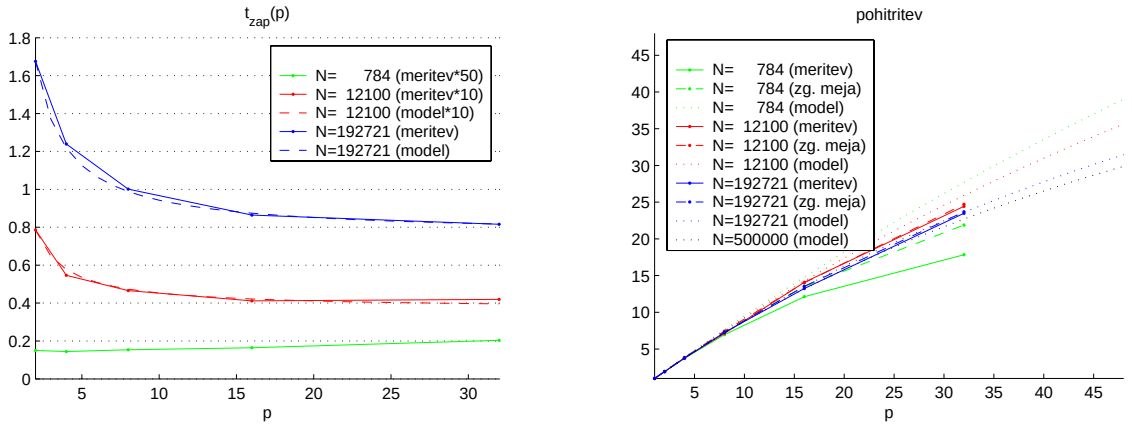
Skupni čas izvajanja in pohitritev

Modela $\hat{t}(N,1)$ in $\hat{t}_{zap}(N,p)$ vstavimo v enačbo 6.1, izraze poenostavimo s programskim paketom Mathematica [Mata] in dobimo modela za skupni čas izvajanja in pohitritev:

$$\hat{t}(N,p) = \frac{3.7 \cdot 10^{-4} N + N^{1.15}\left(9.7 \cdot 10^{-7} + 6.4 \cdot 10^{-7} p\right)}{p},$$

$$\hat{S}(N,p) = \frac{Np}{N + N^{1.15}(0.0026 + 0.0017p)}. \tag{6.2}$$

Pri danem N se torej z večanjem števila procesorjev pohitritev asimptotično približuje največji možni, kar se ob upoštevanju odvisnosti t_{zap} od p, prikazane na levem delu slike 6.7, ujema z enačbo 6.1.

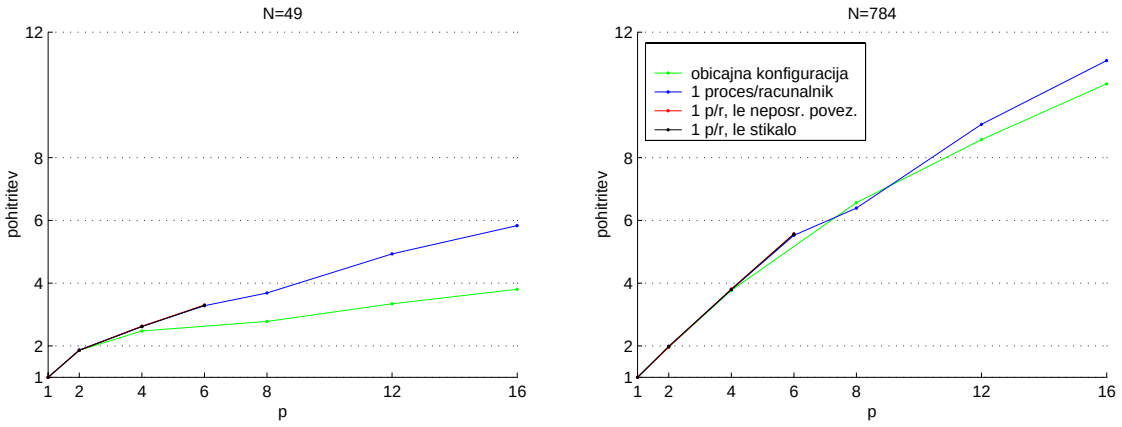


Slika 6.7: Izmerjena odvisnost t_{zap} od p in model (levo) ter primerjava izmerjene pohitritve, njene zgornje meje in napovedi modela (desno).

Ujemanje modela z meritvami je prikazano na desnem delu slike 6.7. Kvalitativno model deluje pravilno za 12 100 in več vozlišč – pohitritev raste s p, vendar vse počasneje, in rahlo pada z N. Za 12 100 vozlišč je napovedana pohitritev precej previsoka, za 192 721 in 500 000 vozlišč pa verjetno drži tudi za $p > 32$. Za majhne N je model kvalitativno napačen. Napovedana pohitritev namreč monotono raste, ko N pada, kar pa pri majhnem številu vozlišč ni več res. Nasploh je uporabnost modela omejena, saj ne upošteva vseh podrobnosti, ki vplivajo na hitrost izvajanja vzporednega programa (izraba predpomnilnika, izraba vodila, način komunikacije, izraba aritmetičnih enot, dogajanje, povezano z operacijskim sistemom itd.).

6.3.4 Vpliv vrste povezovalnega omrežja in hitrosti procesorjev

Slika 6.8 prikazuje vpliv vrste povezav na pohitritev za $N = 49$ (levo) in $N = 784$ (desno) z enodimenzionalno delitvijo. Zelena krivulja so pohitritve z desnega grafa s slike 6.5. Modro so prikazane pohitritve z uporabo le enega procesorja v vsakem izmed 16 računalnikov. Možna razloga za višje pohitritve pri konfiguraciji z enim procesorjem sta morebitno neugodno dogajanje znotraj računalnika (slabše delovanje knjižnice LAM/MPI z dvema procesoma na enem računalniku, konflikti pri dostopu do pomnilnika, slabo razporejanje izjemno kratkih opravil med procesorja ipd.) in dejstvo, da si oba procesorja delita mrežne kartice in zato ne moreta komunicirati hkrati s polno hitrostjo.



Slika 6.8: Vpliv vrste povezav med računalniki na pohitritev.

Program smo izvajali še na manjših delih skupka, ki omogočajo specifične načine komunikacije. Rdeča krivulja na sliki 6.8 je dobljena z uporabo računalnikov kocka1 ter z njo neposredno povezanih kocka2, kocka5, kocka4, kocka13 in zelda. Kocka1 lahko v tem primeru komunicira z vsemi ostalimi hkrati s polno hitrostjo, ob upoštevanju, da je njeno sistemsko vodilo dovolj hitro. Črna krivulja pa je dobljena z računalnikom kocka1 ter kocka10, kocka11, kocka7, kocka12 in kocka15, ki so vsi oddaljeni od kocke1 za 3 ali 4 skoke, torej vsa komunikacija poteka preko omrežnega stikala. Kocka1 je s stikalom povezana le z eno mrežno kartico, zato se pri hkratni komunikaciji z več sosedi hitrost komunikacije s vsakim posameznim sosedom ustrezno zmanjša. Ker se rdeča in črna krivulja skoraj povsem pokrivata z modro, sklepamo, da gola hitrost komunikacijskih povezav nima odločilnega vpliva. Slabe pohitritve z uporabo dveh procesorjev na računalniku so torej posledica dogajanja znotraj računalnika.

S poskusi smo se prepričali, da pri večjem številu vozlišč razlike v času izvajanja na eno- ali dvoprocesorskem računalniku skoraj ni več, saj sedaj prevladuje čas, potreben za računanje, komunikacijski čas pa je manj pomemben. Zanimivo bi bilo preizkusiti tudi komunikacijo izključno preko stikala pri uporabi vseh računalnikov, kjer bi morda stikalo že postalo ozko grlo in s tem zaustavilo izvajanje vzporednega programa. Za

takšen poskus bi bilo potrebno začasno spremeniti usmerjevalne tabele vseh računalnikov v skupku [RŠT04], česar pa v okviru našega dela nismo naredili.

Ocena časa za pošiljanje sporočil

Pošiljanje sporočila dolžine l bajtov traja $\hat{t}(l) = t_s + t_b \cdot l$ časa, kjer je t_s *vzpostavitveni čas* (angl. setup time), t_b pa čas za pošiljanje enega bajta. Meritve hitrosti komunikacije v testnem skupku [RTŠ05a] so za naš skupek pokazale, da je $t_s = 25\mu$s za komunikacijo med neposredno povezanima računalnikoma in $t_s = 63\mu$s za komunikacijo preko stikala. V obeh primerih je testni skupek pri velikih sporočilih dosegel hitrost okrog 600 Mb/s (75 MB/s), torej je $t_b = 0.013\mu$s.

Med izvajanjem vzporedne različice MPM proces, ki generira vozlišča, pošlje vsakemu izmed ostalih po 3 sporočila:

1. oglišča dodeljene poddomene (32 bajtov, $\hat{t}_1 = 25.5\mu$s),
2. število vozlišč v poddomeni in okolici, ki bodo poslana v tretjem sporočilu (4 bajti, $\hat{t}_2 = 25\mu$s),
3. vozlišča ($48N/p$ bajtov).

Pošiljanje vseh treh sporočil v primeru, ko so ta največja ($N = 192\,721, p = 2$), traja največ $t_1+t_2+t_3 = 0.061$ s. Z levega grafa na sliki 6.7 vidimo, da je celotni $t_{zap}(192\,721, 2)$ približno 1.7 s, zato je jasno, da se razlike v hitrosti komunikacije ne morejo poznati na izmerjeni pohitritvi.

Seveda časa komunikacije tudi ni smiselno upoštevati v modelu pohitritve, ki je za takšne vplive pregrob. Upoštevati bi ga morali le, če bi bile povezave bistveno počasnejše. Z uporabo mrežnih kartic hitrosti 100 Mb/s bi bil čas komunikacije približno desetkrat večji, kar bi že opazno vplivalo na pohitritev, vendar se tako počasne povezave danes skoraj ne uporabljajo več.

Vpliv hitrosti procesorjev

Trajanje vseh delov vzporednega programa je torej povečini odvisno od hitrosti procesorja, pri manjšem številu vozlišč (manj kot 100 na procesor) pa tudi od ostalega dogajanja znotraj računalnika. Uporaba drugačnih procesorjev lahko nekoliko spremeni razmerja med časi posameznih gradnikov MPM, česar zaradi množice različnih procesorskih arhitektur ne moremo natančno predvideti. Kljub temu lahko predvidevamo, da se bodo z uporabo za nek faktor hitrejših procesorjev časi $t(N, p)$ skrajšali za podoben faktor, medtem ko se pohitritve ne bodo bistveno spremenile.
Sklep: Pohitritve omejuje zaporedni del programa, čas komunikacije pa ni bistven. Razmerje med hitrostjo povezav in hitrostjo procesorjev zato ne vpliva na pohitritev.

Poglavje 7

Sklep

Mreže proste metode za reševanje parcialnih diferencialnih enačb, ki se razvijajo zadnjih petnajst let, poskušajo odpraviti slabosti metod končnih razlik in končnih elementov, ki dandanes prevladujeta v komercialnih programih za računalniške simulacije in kljub zrelosti doživljata tudi nadaljnje raziskave.

Osnovna ideja mreže prostih metod je definicija baznih funkcij s pomočjo bližnjih vozlišč, ki za razliko od metode končnih elementov niso povezana v mrežo. Tak postopek prinese bistvene razlike v matematični formulaciji, računalniški izvedbi in paralelizaciji metode. V zadnjih desetih letih so se razvile številne različice mreže prostih metod, bolj ali manj uporabne za različne vrste problemov.

V literaturi poleg osnovnih idej najdemo predvsem številna poročila o uspešnem reševanju različnih diferencialnih enačb z mreže prostimi metodami, medtem ko ostaja mnogo odprtih vprašanj tako s strani matematične formulacije in pravilne izbire parametrov kot tudi z računalniške strani, na primer o izvedbi posamičnih gradnikov metod in njihovi paralelizaciji. Nekatera izmed teh vprašanj si prizadeva rešiti ta doktorska disertacija.

Podrobno smo prikazali pretvorbo difuzijske enačbe v sistem linearnih enačb z vsemi tremi metodami. V primerjavi z obema starejšima metodama zahtevajo mreže proste metode več truda pri prvi formulaciji rešitve enačbe, pri čemer moramo posebej izpostaviti določanje nosilne domene točk in zagotavljanje zveznosti rešitve. Trud je poplačan z večjo natančnostjo in možnostjo enostavnega spreminjanja oblike domene. V delu smo se omejili na obetavno inačico mreže proste metode MLPG5 z baznimi funkcijami, dobljenimi z aproksimacijo nad lokalnimi vozlišči. Pokazali smo, da lahko s primerno izbiro parametrov dobimo rešitev podobne natančnosti, kot bi jo dobili z metodo končnih elementov, če bi pri slednji uporabili kar dva- do štirikrat več vozlišč.

Analiza časovne zahtevnosti je pokazala približno desetkrat večjo zahtevnost gradnje linearnega sistema z mreže prostimi metodami kot z metodo končnih elementov. Kljub temu bodo, ko dosežejo večjo zrelost, mreže proste metode bolj priporočljive. Metoda

končnih elementov namreč vključuje tudi časovno zahtevno generiranje mreže in potrebo po človeškem delu, ki v primerjavi z računalniškim časom postaja vse dražje. Pokazali smo tudi, da so mreže proste metode primerne za uporabo na vzporednih računalnikih, kar je glede na precejšnjo časovno zahtevnost potreben pogoj za njihovo širšo uporabo v zahtevnejših problemih.

7.1 Izvirni prispevki k znanosti

Celotna doktorska disertacija, predvsem pa njena druga polovica, je posvečena obravnavi nekaterih odprtih vprašanj na področju primernosti, računalniške izvedbe, časovne zahtevnosti in paralelizacije mreže prostih metod. Naše odgovore in spoznanja lahko strnemo v naslednjih točkah.

Numerične metode za reševanje parcialnih diferencialnih enačb in njihova uporaba na difuzijski enačbi (pregled trenutnega stanja področja)
Predstavili smo tri primerjane metode s stališča teoretičnih osnov ter njihovih prednosti in slabosti posameznih metod. Za vsako izmed treh metod smo prikazali celoten postopek pretvorbe parcialne diferencialne enačbe v sistem navadnih enačb na primeru difuzijske enačbe, ki je zaradi časovne odvisnosti zahtevnejša in v literaturi manj obdelana od časovno neodvisnih enačb. Na kratko smo opisali nekatere metode za reševanje dobljenega sistema linearnih enačb.

Vsiljevanje zveznosti rešitve in določanje povprečne razdalje med vozlišči
Opozorili smo na problem nezveznosti baznih funkcij MLS, kot so definirane v [Liu03]. Predlagali smo način vsiljevanja zveznosti baznih funkcij s pomočjo interpolacije povprečne razdalje med vozlišči. Za tako definirane zvezne bazne funkcije smo pokazali enako ali večjo natančnost rešitve difuzijske enačbe kot pri nezveznih funkcijah.

Poleg tega smo predlagali dva postopka določanja velikosti nosilne domene: NumIdCorr (idealno število nosilnih vozlišč s popravkom) in AvgId (povprečna razdalja med idealnim številom vozlišč) – slednji je podoben postopku, približno opisanemu v [Liu03]. Poskusi so pokazali, da je ustreznejši postopek AvgId.

Primernost metod za reševanje difuzijske enačbe in izbira parametrov
Vse tri metode smo prototipno implementirali in analizirali, kako ločljivost prostorske diskretizacije in časovni korak vplivata na natančnost. Za mreže proste metode smo analizirali tudi vpliv ostalih parametrov in poiskali vrednosti, optimalne za reševanje difuzijske enačbe. Natančnost vseh treh metod smo primerjali na dveh testnih primerih.

Izpopolnjena izvedba mreže proste metode
Predlagali smo izvedbo mreže proste metode, ki vozlišča hrani v k-D drevesu, kar

omogoča učinkovito iskanje vozlišč v nosilni domeni. Podrobno smo opisali tudi izvedbo ostalih gradnikov metode in jo implementirali v višjem programskem jeziku.

Analiza časovne in prostorske zahtevnosti
Na podlagi predlagane izvedbe smo podrobno teoretično in eksperimentalno analizirali zahtevnost mreže proste metode in njenih posameznih gradnikov. Primerjava z zahtevnostjo metod končnih razlik in končnih elementov nam je dala dodaten vpogled v primernost vseh treh metod.

Vzporedna izvedba mreže proste metode
Predlagali smo dva načina delitve domene v vzporedni izvedbi mreže proste metode. V nasprotju z načinom, opisanim na primer v [DUAL00], predlagana načina ne slonita na za končne elemente običajni delitvi grafa.

Enodimenzionalna delitev domeno razreže na rezine, hierarhična pa enako kot k-D drevo vozlišča najprej razdeli na levo in desno polovico, nato vsako izmed njiju na spodnjo in zgornjo in tako naprej, dokler ni število delov enako številu procesorjev. Oba načina zagotavljata enakomerno obremenitev vseh procesorjev, vendar je pri hierarhični delitvi začetni zaporedni del programa nekoliko daljši. Paralelizacijo z opisanima načinoma delitve domene smo primerjali z znanimi načini paralelizacije metod končnih razlik in končnih elementov s stališča primernosti za reševanje dobljenega linearnega sistema.

Analiza pohitritve mreže proste metode na računalniškem skupku
Vzporedno izvedbo mreže proste metode z obema načinoma delitve domene smo testirali na računalniškem skupku in primerjali pohitritve. S pomočjo meritve posameznih delov programa smo analizirali vplive parametrov na pohitritve in ocenili obnašanje programa na vzporednem računalniku z drugačno hitrostjo procesorjev in povezav med njimi.

7.2 Smernice za nadaljnje delo

Vsako obsežnejše delo, ki tako kot pričujoča disertacija obravnava manj zrelo področje, pušča mnoga vprašanja vnemar in odpira tudi nekatera nova. Nekatera želimo še posebej izpostaviti.

Dasiravno predlagani način vsiljevanja zveznosti baznih funkcij dobro deluje, uvaja nov parameter l in tudi z matematičnega stališča ni najbolj eleganten. Zato bi naše neuspešne poskuse iskanja boljše rešitve veljalo nadaljevati. Poleg tega bi lahko poizkusili poenostaviti postopek določanja velikosti nosilne domene. Morda je možno tudi izpeljati odvode baznih funkcij MLS s Householderjevo metodo, čeprav bi morali očitno ubrati drugo pot kot z metodo normalnih enačb.

Metoda končnih razlik se pogosto uporablja z eksplicitno časovno integracijo, ki namesto reševanja sistema v vsakem koraku zahteva zgolj množenje matrike z vektorjem. Primerjava časovne zahtevnosti metod bi bila popolnejša, če bi vključili tudi to možnost, pri čemer bi seveda upoštevali krajši časovni korak, potreben pri eksplicitni integraciji.

Nadaljnji razvoj zahteva prilagajanje metode multigrid za reševanje sistemov, dobljenih z mreže prostimi metodami. Nekateri v zadnjem času objavljeni rezultati so opogumljajoči, vendar zaenkrat uporabni samo za posamezne različice mreže prostih metod.

Primerjavo natančnosti metod bi bilo koristno ponoviti na kaki drugi diferencialni enačbi, predvsem pa tudi v treh dimenzijah. Tudi analiza časovne zahtevnosti za tridimenzionalne probleme bi prišla prav.

Končno bi kazalo preveriti tudi možnosti izboljšanja vzporedne izvedbe programa, ki bi se verjetno lahko še nekoliko bolj približala linearni pohitritvi, in izmeriti pohitritve na vzporednih računalnikih z več procesorji in drugačno arhitekturo povezovalnega omrežja.

Literatura

[AAE03] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *Journal of Computer and System Sciences*, 66:207–243, 2003.

[AG94] G. S. Almasi and A. Gottlieb. *Highly Parallel Computing*. Benjamin-Cummings, Redwood City, 2nd edition, 1994.

[AGG02] Pankaj K. Agarwal, Jie Gao, and Leonidas J. Guibas. Kinetic medians and kd-trees. In *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*, pages 5–16. Springer-Verlag, 2002.

[ALU] Alubook – lexical knowledge about aluminium. `http://www.alu-info.dk/Html/alulib/modul/albook40.htm`.

[AS76] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. Dover, New York, 1976.

[AS02] S. N. Atluri and S. Shen. *The Meshless Local Petrov-Galerkin (MLPG) Method*. Tech Science Press, Encino, 2002.

[AŠT02] V. Avbelj, M. Šterk, and R. Trobec. Lead selection in body surface electrocardiography by exhaustive search optimisation. In *Parallel Numerics '02, Theory and Applications*, pages 201–207. Jožef Stefan Institute and University of Salzburg, 2002.

[Avb03] V. Avbelj. *Analiza električne aktivnosti srca s pomočjo večkanalne elektrokardiografije*. Doktorska disertacija, Fakulteta za elektrotehniko, Univerza v Ljubljani, 2003.

[BBC+94] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Society for Industrial and Applied Mathematics, Philadelphia, 1994.

[BCW75] H. F. Bowman, E. G. Cravalho, and M. Woods. Theory, measurement, and application of thermal properties of biomaterials. *Annual Review Biophysics and Bioengineering*, 4:43–80, 1975.

[BE95] M. W. Bern and D. Eppstein. Mesh generation and optimal triangulation. In Ding-Zhu Du and Frank Kwang-Ming Hwang, editors, *Computing in Euclidean Geometry*, Lecture Notes Series on Computing, pages 47–123. World Scientific, 1995.

[Bic] Biconjugate gradient stabilized method – from MathWorld. `http://mathworld.wolfram.com/BiconjugateGradientStabilizedMethod.html`.

[BKO+96] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl. Meshless methods: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering, special issue on Meshless Methods*, 139:3–47, 1996.

[BKOS00] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry – Algorithms and Applications*. Springer, Berlin, 2nd edition, 2000.

[BLA] BLAS (basic linear algebra subprograms) home page. `http://www.netlib.org/blas/`.

[BLG94] T. Belytschko, Y. Y. Lu, and L. Gu. Element-free galerkin methods. *International Journal for Numerical Methods in Engineering*, 37:229–256, 1994.

[Buc95] G. R. Buchanan. *Theory and Problems of Finite Element Analysis*. Schaum's Outline Series. McGraw-Hill, New York, 1995.

[CJ59] H. S. Carslaw and J. C. Jaeger. *Conduction of Heat in Solids*. Oxford University Press, London, 1959.

[Cle93] W. S. Cleveland. *Visualising Data*. Hobart Press, Summit, New Jersey, 1993.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, 2nd edition, 2001.

[Com] Comsol multiphysics (naslednik paketa femlab). `http://www.comsol.com/products/multiphysics/`.

[Del34] B. N. Delaunay. Sur la sphere vide. *Izvestia akademii nauk SSSR*, 7:793–800, 1934.

[DHL03] C. C. Douglas, G. Haase, and U. Langer. *A tutorial on elliptic PDE solvers and their parallelization*. Society for Industrial and Applied Mathematics, Philadelphia, 2003.

[DUAL00] K. T. Danielson, R. A. Uras, M. D. Adley, and S. Li. Large-scale application of some modern CSM methodologies by parallel computation. *Advances in Engineering Software*, 31:501–509, 2000.

[FBF77] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.

[FJL+88] G. C. Fox, M. A. Johnsson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker. *Solving Problems on Concurrent Processors, Volume 1: General Techniques and Regular Problems*. Prentice-Hall International, Englewood Cliffs, 1988.

[Fle88] C. A. J. Fletcher. *Computational Techniques for Fluid Dynamics*. Springer Verlag, Berlin, 1988.

[FM04] T. P. Fries and H. G. Matthies. Classification and overview of meshfree methods. Technical report, Technische Universität Braunschweig, 2004.

[GC98] M. A. Golberg and C. S. Chen. The method of fundamental solutions for potential, helmholtz and diffusion problems. In M.A. Golberg, editor, *Boundary Integral Methods - Numerical and Mathematical Aspects*, pages 103–176. Computational Mechanics Publications, 1998.

[GDT+] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. GNU scientific library: Reference manual for GSL version 1.7. `http://www.gnu.org/software/gsl/manual/gsl-ref_toc.html`.

[GGKK03] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison-Wesley, New York, 2nd edition, 2003.

[GMS92] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, 13:333–356, 1992.

[GO93] G. Golub and J. M. Ortega. *Scientific Computing - An Introduction with Parallel Computing*. Academic Press Inc., Boston, 1993.

[Haa98] G. Haase. Parallel incomplete Cholesky preconditioners based on the non-overlapping data distribution. *Parallel Computing*, 24:1685–1703, 1998.

[HE88] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. Institute of Physics Publishing, Bristol, 1988.

[Hea02] M. T. Heath. *Scientific Computing: An Introductory Survey*. WCB/McGraw-Hill, New York, 2nd edition, 2002.

[ICT04] M. S. Ingber, C. S. Chen, and J. A. Tanski. A mesh free approach using radial basis functions and parallel domain decomposition for solving three-dimensional diffusion equations. *International Journal for Numerical Methods in Engineering*, 60:2183–2201, 2004.

[IOCP03] S. R. Idelsohn, E. Onate, N. Calvo, and F. Del Pin. The meshless finite element method. *International Journal for Numerical Methods in Engineering*, 58:893–912, 2003.

[Kat03] P. I. Kattan. *MATLAB Guide to Finite Elements: an Interactive Approach*. Springer-Verlag, Berlin, 2003.

[KGT99] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *Proceedings of the 18th ACM Symposium on Principles of Database Systems*, pages 261–272, 1999.

[KK98] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20:359–392, 1998.

[Kod00] D. Kodek. *Arhitektura računalniških sistemov*. BiTim, Ljubljana, 2000.

[KŠR02] P. Korošec, J. Šilc, and B. Robič. An ant-colony-optimization approach to the mesh partitioning problem. In *Parallel Numerics '02, Theory and Applications*, pages 123–132. Jožef Stefan Institute and University of Salzburg, 2002.

[LA00] H. Lin and S. N. Atluri. Meshless local Petrov-Galerkin (MLPG) method for convection-diffusion problems. *Computer Modeling in Engineering & Sciences*, 2:45–60, 2000.

[LAJ93] W. K. Liu, J. Adee, and S. Jun. Reproducing kernel and wavelet particle methods for elastic and plastic problems. In D. J. Benson, editor, *Advanced Computational Methods for Material Modeling*, pages 175–190. ASME, 1993.

[LAM] LAM/MPI home page. `http://www.lam-mpi.org/`.

[LG03] G. R. Liu and Y. T. Gu. A matrix triangularization algorithm for the polynomial point interpolation method. *Computer Methods in Applied Mechanics and Engineering*, 192:2269–2295, 2003.

[LHKK79] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, 5:308–323, 1979.

[Liu03] G. R. Liu. *Mesh Free Methods: Moving beyond the Finite Element Method*. CRC Press, Boca Raton, 2003.

[LOS04] K. H. Leem, S. Oliveira, and D. Stewart. Algebraic multigrid (AMG) for saddle point systems from meshfree discretizations. *Numerical Linear Algebra and Applications*, 11:293–308, 2004.

[LS81] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of Computation*, 37:141–158, 1981.

[Luc77] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82:1013–1024, 1977.

[Mata] Mathematica: The way the world calculates. `http://www.wolfram.com/products/mathematica/index.html`.

[Matb] MathWorks: Matlab documentation. `http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html`.

[Mey00] C. D. Meyer. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, 2000.

[MPI] MPICH home page. `http://www-unix.mcs.anl.gov/mpi/mpich/`.

[MR00] G. C. Mai and C. A. F. De Rose. Low cost cluster architectures for parallel and distributed processing. *CLEI Electronic Journal*, 3:1–9, 2000.

[NO99] F. K. Nani and M. N. Oguztöreli. Modelling and simulation of chemotherapy of haematological and gynaecological cancers. *IMA Journal of Mathematics Applied in Medicine and Biology*, 16:39–91, 1999.

[Ö94] M. Necati Özisik. *Finite Difference Methods in Heat Transfer*. CRC Press, Boca Raton, 1994.

[Ore97] B. Orel. *Osnove numerične matematike*. Založba FE in FRI, Ljubljana, 1997.

[Owe98] S. J. Owen. A survey of unstructured mesh generation technology. In *Proceedings of 7th International Meshing Roundtable*, pages 239–267. Sandia National Laboratories, 1998.

[PHB+00] J. B. Pormann, C. S. Henriquez, J. A. Board, D. J. Rose, D. M. Harrild, and A. P. Henriquez. Computer simulations of cardiac electrophysiology. In *Proceedings of the EC2000 Conference*, pages 56–57, 2000.

[Pip00] G. Pipan. Vzporedno računanje na gručah računalnikov. Magistrska naloga, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2000.

[Pol64] G. B. Pollard. *Lectures on Partial Differential Equations*. Wiley, New York, 1964.

[PŠT04] M. Praprotnik, M. Šterk, and R. Trobec. Inhomogeneous heat-conduction problems solved by a new explicit finite difference scheme. *International Journal of Pure and Applied Mathematics*, 13:275–291, 2004.

[RŠT04] I. Rozman, M. Šterk, and R. Trobec. Strojna sestava in programske nastavitve za skupek računalnikov. Tehnično poročilo, Institut Jožef Stefan, 2004.

[RTŠ05a] I. Rozman, R. Trobec, and M. Šterk. Hitrost komunikacije v LAM/MPI in MPICH okoljih. V *Zborniku štirinajste elektrotehniške in računalniške konference ERK 2005*, str. 3–6. Slovenska sekcija IEEE, 2005.

[RTŠ05b] I. Rozman, R. Trobec, and M. Šterk. Tuning communication in gigabit ethernet cluster. In *Parallel Numerics '05 : Theory and Applications*, pages 207–216. Jožef Stefan Institute and University of Salzburg, 2005.

[Sch03] A. W. M. van Schijndel. Modeling and solving building physics problems with FemLab. *Building and Environment*, 38:319–327, 2003.

[SOHL+96] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI - The Complete Reference*. The MIT Press, Cambridge, 1996.

[SR97] L. F. Shampine and M. W. Reichelt. The Matlab ODE suite. *SIAM Journal on Scientific Computing*, 18:1–22, 1997.

[Str97] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Boston, 3rd edition, 1997.

[Sur] M. D. Suresh. Blitz++ with VTx library. `http://mdsuresh.tripod.com/blitz-0.6.x.zip`.

[SZ02] C. Shen and J. Zhang. Parallel two level block ILU preconditioning techniques for solving large sparse linear systems. *Parallel Computing*, 28:1451–1475, 2002.

[ŠT03] M. Šterk and R. Trobec. Parallel performance of a multigrid poisson solver. In *Proceedings of Second International Symposium on Parallel and Distributed Computing*, pages 238–243. IEEE Computer Society, 2003.

[ŠT04] M. Šterk and R. Trobec. A multigrid poisson solver on general 3-dimensional domains. In *Parallel processing and applied mathematics (Lecture notes in computer science, 3019)*, pages 1052–1058. Springer Verlag, 2004.

[TSGG98] R. Trobec, B. Slivnik, B. Geršak, and T. Gabrijelčič. Computer simulation and spatial modelling in heart surgery. *Computers in Biology and Medicine*, 4:393–403, 1998.

[Vel] T. Veldhuizen. Blitz++ user's guide. `http://www.oonumerics.org/blitz/manual/Frames.html`.

[Ver04] O. Verdier. Benchmark of femlab, fluent and ansys. Technical report, Lund Institute of Technology, 2004.

[VJ97] T. L. Veldhuizen and M. E. Jernigan. Will C++ be faster than Fortran? In *ISCOPE '97: Proceedings of the Scientific Computing in Object-Oriented Parallel Environments*, Lecture Notes in Computer Science, pages 49–56. Springer-Verlag, 1997.

[War83] F. W. Warner. *Foundations of differentiable manifolds and Lie groups*. Springer-Verlag, New York, 1983.

[Wen98] H. Wendland. Error estimates for interpolation by compactly supported radial basis functions of minimal degree. *Journal of Approximation Theory*, 93:258–272, 1998.

[Wes91] P. Wesseling. *An Introduction to Multigrid Methods*. Wiley, New York, 1991.

[WLSO01] D. R. White, R. W. Leland, S. Saigal, and S. J. Owen. The meshing complexity of a solid: An introduction. In *Proceedings of 10th International Meshing Roundtable*, pages 373–384. Sandia National Laboratories, 2001.

[Wol87] M. Wolfe. Iteration space tiling for memory hierarchies. In *Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing*, pages 357–361. Society for Industrial and Applied Mathematics, 1987.

Stvarno kazalo

Seznami kratic, oznak in prevodov

kratica	stran	pomen
BLAS	97	Basic Linear Algebra Subprograms
DE	3	diferencialna enačba
EFG	10	element free Galerkin method
EPG	10	elementov prosta metoda Galerkina
FDM	25	finite difference method
FEM	26	finite element method
GSL	97	GNU Scientific Library
iščiKvad	99	različica algoritma za numerično integracijo v izvedbi MPM
iščiOsn	99	različica algoritma za numerično integracijo v izvedbi MPM
MFLOPS	97	million of floating point operations per second
MKE	26	metoda končnih elementov
MKR	25	metoda končnih razlik
MLPG	10	mreže prosta lokalna Petrov-Galerkinova metoda
MLPG1-6	55	različice MLPG
MLS	40	moving least squares
MPI	12	Message Passing Interface
MPM	9	mreže prosta metoda, mreže proste metode
MPM1-2	64	MPM z aproksimacijo MLS z monomi stopenj do 1 oziroma do 2
nAvgId	78	nezvezne bazne funkcije MLS, kjer je d_S določena s postopkom AvgId
NDE	3	navadna diferencialna enačba
nNumId	78	nezvezne bazne f. MLS, kjer je d_S določena s postopkom NumIdCorr
PDE	4	parcialna diferencialna enačba
PIM	54	point interpolation method
SMP	11	symmetric multiprocessor
SOR	34	successive over-relaxation
zAvgId	78	zvezne bazne funkcije MLS, kjer je d_S določena s postopkom AvgId
zNumId	78	zvezne bazne funkcije MLS, kjer je d_S določena s postopkom NumIdCorr

oznaka	stran	pomen
∇	19	nabla, operator parcialnega odvajanja
α_Q	68	brezdimenzijska velikost kvadraturne domene
α_S	49	brezdimenzijska velikost nosilne domene
Γ	4	rob domene Ω
Γ_Q	56	rob kvadraturne domene Ω_Q
$\Gamma_{Q,bound}$	59	del Γ_Q, ki sovpada z globalnim robom Γ
$\Gamma_{Q,int}$	59	del Γ_Q, ki ne sovpada z globalnim robom Γ
Δx	16	ločljivost prostorske diskretizacije
Δt	22	časovni korak
λ	19	koeficient toplotne prevodnosti
	23	lastna vrednost
ρ	19	gostota
$\rho(A)$	23	spektralni radij matrike
$\phi, \mathbf{\Phi}$	8	bazna funkcija, vektor baznih funkcij
Ω	4	domena, na kateri rešujemo PDE
Ω_P	120	poddomena, nastala z delitvijo Ω med procesorje
Ω_Q	55	kvadraturna domena
Ω_S	59	nosilna domena
Ω_{SQ}	60	nosilna domena kvadraturne domene
A	26	sistemska matrika linearnega sistema, dobljenega z diskretizacijo PDE
	46	sistemska matrika za izračun koeficientov aproksimacije MLS
A_Ω	95	ploščina domene Ω
A_S	49	ploščina nosilne domene
$\mathbf{a}$	41	vektor koeficientov za interpolacijo ali aproksimacijo v MPM
B	26	matrika desnih strani linearnega sist., dobljenega z diskretizacijo PDE
	46	matrika desnih strani za izračun koeficientov aproksimacije MLS
C	31	matrika dušenja
C^{int}	60	matrika dušenja za notranja vozlišča MPM
C^L	87	lokalna matrika dušenja
c	19	difuzijska konstanta
c_p	19	specifična toplota
$condest(A)$	113	ocena števila občutljivosti matrike A z Matlabovo funkcijo `condest`
$\overline{D}$	95	matrika povprečnih razdalj $\overline{d}(\mathbf{x})$, ovrednotenih na pravilni mreži
D_S	49	velikost (premer) vzorčne domene
$\overline{d}(\mathbf{x})$	49	povprečna razdalja med vozlišči v okolici točke $\mathbf{x}$
$\overline{d}^*$	95	povprečna razdalja med vozlišči v celotni domeni Ω
d_{max}	120	zgornja meja $\overline{d}$
$d_S(\mathbf{x})$	45	velikost (premer) nosilne domene točke $\mathbf{x}$
d_{SQ}	100	velikost (premer) Ω_{SQ}
e	86	prostor (število bajtov) za hranjenje enega števila v plavajoči vejici
F	3	funkcija v implicitnem zapisu diferencialne enačbe $F = 0$
f	3	funkcija desne strani v eksplicitnem zapisu DE
	100	razmerje površin Ω_{SQ} in Ω_S

oznaka	stran	pomen
$\mathbf{f}$	26	vektor sil
g	19	funkcija izvorov v difuzijski enačbi
$\mathbf{g}$	46	pomožni vektor za formulacijo odvodov baznih funkcij MLS
I	26	enotska matrika
J	22	Jacobijeva matrika
K	29	togostna matrika
K^{int}	60	togostna matrika za notranja vozlišča MPM
K^L	30	lokalna togostna matrika
L'	112	faktor, dobljen z nepopolnim LU razcepom
l	95	velikost pravilne mreže, na kateri ovrednotimo $\overline{d}$
m	44	število funkcij v vektorju $\mathbf{p}$
N	15	število vozlišč
N_Γ	99	število notranjih vozlišč MPM, katerih kvadraturna domena seka Γ
$\mathbf{n}$	18	zunanji normalni vektor na rob
n_G	99	število kvadraturnih točk v vsaki dimenziji
n_I	70	idealno število nosilnih vozlišč
n_x	18	komponenta $\mathbf{n}$ v smeri x
$nnz(A)$	33	število neničelnih elementov matrike A
P	42	momentna matrika
P_W	46	utežena momentna matrika
p	11	število procesorjev
$\mathbf{p}$	41	vektor funkcij (običajno monomov), uporabljenih za interpolacijo ali aproksimacijo v MPM
$\mathbf{r}, r$	6	ostanek
S	126	pohitritev
S_M	87	prostorska zahtevnost metode M
S_{max}	124	zgornja meja pohitritve vzporednega programa za MPM
s	88	povprečno število sosedov vozlišča v MKE
T_M	86	časovna zahtevnost metode M
t	125	čas izvajanja vzporednega programa za MPM
t_b	128	čas za pošiljanje enega bajta
t_{min}	124	spodnja meja časa izvajanja vzporednega programa za MPM
t_s	128	vzpostavitveni čas
t_{zap}	124	čas izvajanja začetnega zaporednega dela vzporednega progr. za MPM
$\mathbf{u}$	8	vektor vozliščnih parametrov
$\hat{u}, \hat{\mathbf{u}}$	6	poskusna funkcija, približek rešitve
u_i	28	vozliščni parameter vozlišča i
$\mathbf{u}^{(k)}$	25	vrednost $\mathbf{u}$ v časovnem koraku k
u_x	6	parcialni odvod u po x
W	7	utežna funkcija
$\mathbf{x}_j$	27	vozlišče j
$\mathbf{x}_R$	90	vozlišče, shranjeno v korenu (pod)drevesa
$\mathbf{x}_S$	41	množica nosilnih vozlišč točke $\mathbf{x}$

tuj izraz	prevod
background mesh	mreža elementov v ozadju
base function	bazna funkcija
cluster	skupek, gruča
collocation	kolokacija
condition number	število občutljivosti
conditioning	občutljivost, pogojenost
conjugate gradient squared	kvadratni konjugirani gradienti
conjugate gradients	konjugirani gradienti
damping matrix	matrika dušenja
decomposition	razcep
degree	stopnja
efficiency	učinkovitost
element free Galerkin method	elementov prosta metoda Galerkina
essential boundary condition	bistveni robni pogoj
fill in	zapolnjevanje
finite difference method	metoda končnih razlik
finite element method	metoda končnih elementov
graph partitioning	razdeljevanje grafov
hat function	funkcija v obliki klobuka
heap sort	urejanje s kopico
hop	skok, korak
influence domain	domena vpliva
integration domain	integracijska domena
iteration-space tiling	preurejanje vrstnega reda operacij za čimvečjo verjetnost predpomnilniškega zadetka
Lagrange multiplier	Lagrangeov multiplikator
linear least squares	linearni najmanjši kvadrati
locally weighted regression	lokalno utežena regresija
mesh free methods	mreže proste metode, brezmrežne metode
meshless local boundary integral equation	mreže prosta enačba robnega integrala
meshless local Petrov-Galerkin methods	mreže proste lokalne Petrov-Galerkinove metode
meshless methods	mreže proste metode
Message Passing Interface	vmesnik za izmenjavo sporočil
moving least squares	gibljivi najmanjši kvadrati
multicore chip	večjedrni čip
multiquadrics	multikvadrika
natural boundary condition	naravni robni pogoj
nodal parameter	parameter vozlišča, vozliščni parameter
node	vozlišče
order	red

tuj izraz	prevod
partition tree	razdelitveno drevo
penalty method	kazenska metoda
per partes	po delih
point interpolation methods	interpolacijske mreže proste metode
preconditioning	predpogojevanje
quadrature domain	kvadraturna domena
quicksort	urejanje s porazdelitvami
reproducing kernel particle method	metoda delcev, ki ohranjajo jedro
residual	ostanek
restriction	omejevanje
shared memory	skupni pomnilnik
smoothed particle hydrodynamics	hidrodinamika z zglajenimi delci
smoother	zglajevalka
source function	funkcija izvorov
speedup	pohitritev
stiffness matrix	togostna matrika
support domain	nosilna domena
switch	omrežno stikalo
symmetric multiprocessor	simetrični večprocesorski računalnik
template expressions	šablonski izrazi
trial function	poskusna funkcija
variational principle	variacijsko načelo
weak form	šibka oblika
weight function	utežna funkcija