SVEUČILIŠTE U ZAGREBU
PMF – MATEMATIČKI ODJEL

Sanja Singer i Saša Singer

Paralelni algoritmi 2

Predavanja i vježbe (verzija 1.02)

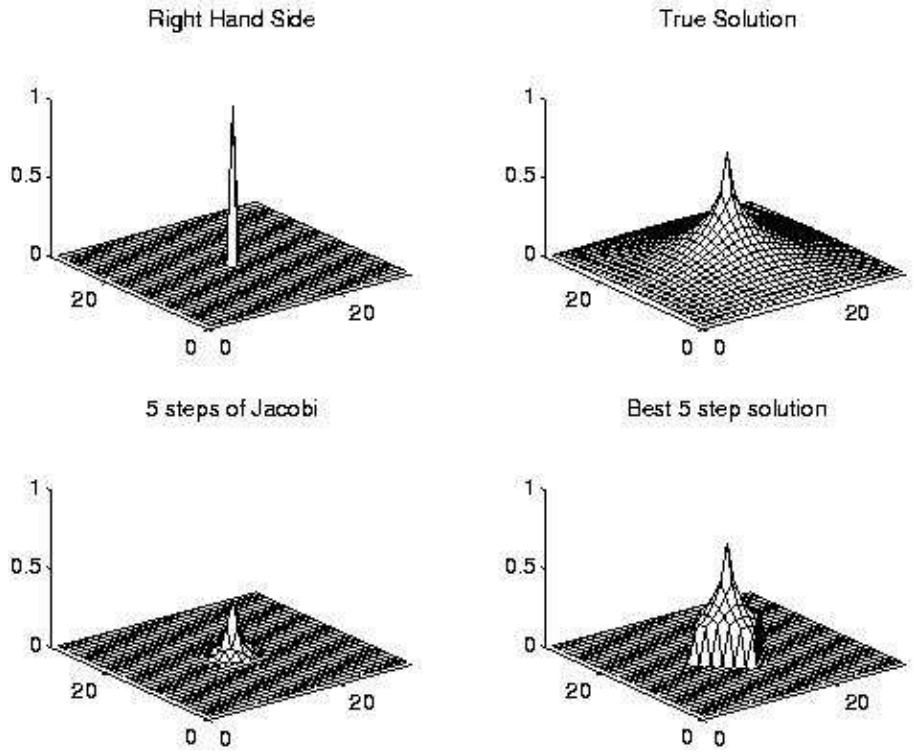
Zagreb, 2003.

Sadržaj

1. Rješavanje diskretne Poissonove jednadžbe Multigrid metodom

Multigrid metode izmišljene su za rješavanje parcijalnih diferencijalnih jednadžbi, poput Poissonove, ali se mogu primijeniti i na širu klasu problema.

Za razliku od standardnih iterativnih metoda za rješavanje pripadnog linearnog sustava, brzina konvergencije multigrid metode **ne ovisi** o veličini N problema, tj. o finoći diskretizacije.



1.1. Uvod u Multigrid

Multigrid je "divide-and-conquer" ("podijeli-pa-vladaj") algoritam za rješavanje diskretne Poissonove jednadžbe. U praksi se često koristi i za druge slične ("eliptičke") parcijalne diferencijalne jednadžbe.

U Multigridu se pristup "podijeli-pa-vladaj" koristi na dva povezana načina. Prvi je "rekurzivan" po gustoći mreže. To znači da se početno rješenje za $n \times n$ mrežu dobiva korištenjem grublje "polovične" $(n/2) \times (n/2)$ mreže kao aproksimacije, uzimanjem svake druge mrežne točke iz $n \times n$ mreže (raspolavljanje u svakoj dimenziji). Grublja $(n/2) \times (n/2)$ mreža opet se aproksimira još grubljom $(n/4) \times (n/4)$ mrežom, i tako redom — rekurzivno.

Ideja "ugrubljavanja" — korištenja grube aproksimacije problema za dobivanje njegovog približnog rješenja, i rekurzivno, korištenje još grublje aproksimacije grubog problema, javlja se u raznim brzim algoritmima. Na primjer, takav pristup opet ćemo susresti kad budemo razmatrali particioniranje grafova.

Ovaj dio "podijeli-pa-vladaj" pristupa odnosi se na **prirodnu domenu** problema, a služi očuvanju globalnosti u svim fazama rješavanja problema, tako da eliminiramo problem sporog širenja lokalnih informacija koji se javlja kod svih klasičnih iterativnih metoda ("izglađivanje lokalnim usrednjavanjem").

Drugi način na koji se "podijeli-pa-vladaj" pristup koristi u Multigridu je u tzv. **domeni frekvencija** ili frekvencijskoj domeni. Za to treba grešku promatrati kao linearnu kombinaciju svojstvenih vektora, ili sinusnih funkcija s različitim frekvencijama. Intuitivno govoreći, posao koji radimo na određenoj mreži treba umanjiti (ili eliminirati) grešku u polovini frekvencijskih komponenti čija greška još nije smanjena (ili eliminirana) na grubljim mrežama.

U stvarnosti, ono što radimo na pojedinoj mreži je usrednjavanje rješenja u svakoj točki mreže obzirom na njezine susjede, varijacijom Jacobijeve metode. Taj postupak "izglađuje" rješenje, što je ekvivalentno uklanjanju visokih frekvencija (brzih oscilacija) u pogrešci. Detaljnija ilustracija ovih pojmova slijedi malo kasnije.

Podrobniji matematički uvod u multigrid algoritam može se naći u knjizi W. Briggs: "A Multigrid Tutorial", SIAM, 1987. Jednostavnu Matlab implementaciju multigrida napravio je Jim Demmel i nalazi se na Web adresi

`http://www.cs.berkeley.edu/~demmel/ma221/Matlab/MG_README.html`

1.2. Pregled Multigrid metode

Počinjemo s opisom algoritma na globalnom nivou, a onda ćemo ga dopuniti potrebnim detaljima. Krenimo od raspolavljanja u domeni. Vidjeli smo da se FFT

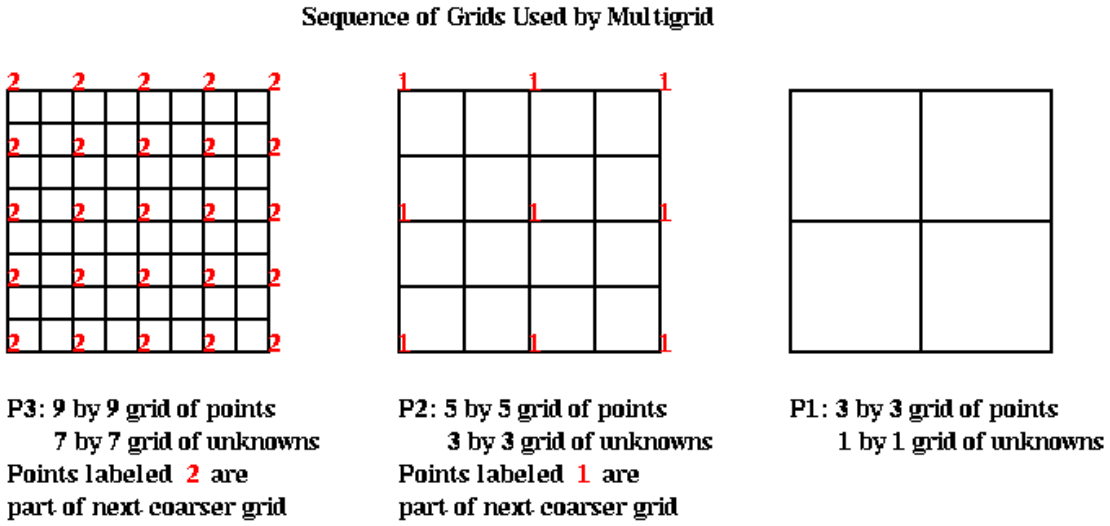
obično koristi na mreži koja ima $2^m \times 2^m$ nepoznanica (u dvije dimenzije), jer to odgovara raspolavljanju broja točaka (komponente vektora).

Za razliku od toga, pokazuje se da je princip "raspolavljanja" u domeni (uzmi svaku drugu točku) zgodnije interpretirati kao raspolavljanje intervala, tako da broj podintervala (a ne točka) treba biti potencija od 2. Zbog toga se Multigrid koristiti na mreži koja ima $(2^m - 1) \times (2^m - 1)$ nepoznanica — to su nepoznate vrijednosti rješenja u unutrašnjim čvorovima mreže.

Kad dodamo rubne čvorove, u kojima su zadane vrijednosti rješenja (rubne vrijednosti su 0 u našem modelnom problemu), onda dobivamo cijelu $(2^m + 1) \times (2^m + 1)$ mrežu na kojoj radi algoritam. Takva mreža odgovara podjeli svake stranice kvadrate na 2^m jednakih podintervala. Označimo $n = 2^m + 1$. Mreže za Multigrid prikazane su na slici malo niže.

Neka je $P(i)$ problem rješavanja diskretne Poissonove jednadžbe na mreži $(2^i + 1) \times (2^i + 1)$ s $(2^i - 1)^2$ nepoznanica. Problem je određen veličinom mreže i, matricom koeficijenata $T(i)$ i desnom stranom $b(i)$.

Mi ćemo generirati niz problema $P(m), P(m-1), P(m-2), \ldots, P(1)$ na sve grubljim i grubljim mrežama, i to tako da je rješenje $P(i-1)$ dobra aproksimacija rješenja $P(i)$.



Neka je $b(i)$ desna strana linearnog sustava za $P(i)$ i neka je $x(i)$ aproksimativno rješenje problema $P(i)$. Tada su $x(i)$ i $b(i)$ polja veličine $(2^i - 1) \times (2^i - 1)$ s vrijednostima u svakoj točki pripadne mreže. Uočite da $x(i)$ **ne mora** biti rješenje linearnog sustava za problem $P(i)$ s desnom stranom $b(i)$, već je dovoljno da $x(i)$ bude neka aproksimacija tog rješenja. Upravo to je jedna od ključnih ideja ove metode, tako da u trenutku kad "rješavamo" $P(i)$, iskoristimo ovo poznato približno rješenje $x(i)$ za dobivanje još boljeg rješenja problema $P(i)$ (koje i opet ne

mora biti egzaktno).

Da bismo objasnili kako algoritam radi, potrebno je uvesti nekoliko operatora koji djeluju na problem i njegovo približno rješenje na nekoj mreži. Ti operatori onda ili poboljšavaju rješenje problema na toj mreži ili sve skupa transformiraju u neki drugi problem (i njegovo približno rješenje) na nekoj drugoj mreži. Možemo zamisliti da ovi operatori rade na parovima oblika $(P(i), x(i))$, koji se sastoje od problema $P(i)$ i njegovog približnog rješenja $x(i)$. Međutim, problem $P(i)$ potpuno je određen svojom desnom stranom $b(i)$, jer se matrica sustava za $P(i)$ neće mijenjati. Zato uzimamo da operatori rade na parovima oblika $(b(i), x(i))$.

- **Operator restrikcije** $R(i)$ uzima par $(b(i), x(i))$, za problem $P(i)$ (određen njegovom desnom stranom $b(i)$) i aproksimativno rješenje $x(i)$, i preslikava ga u par $(b(i-1), x(i-1))$, za jednostavniji problem $P(i-1)$ na sljedećoj grubljoj mreži s početnom aproksimacijom $x(i-1)$:

  $$(b(i-1), x(i-1)) = R(i)(b(i), x(i)).$$

  Vidjet ćemo da se restrikcija jednostavno implementira tako da u svakom čvoru mreže računamo težinsku sredinu vrijednosti u samom čvoru i njegovim najbližim susjedima.

- **Operator interpolacije** $In(i-1)$ uzima aproksimativno rješenje $x(i-1)$ za $P(i-1)$ i pretvara ga u aproksimativno rješenje $x(i)$ za problem $P(i)$ na sljedećoj finijoj mreži:

  $$(b(i), x(i)) = In(i-1)(b(i-1), x(i-1)).$$

  Implementacija tog operatora, također, zahtijeva samo težinsku sredinu s najbližim susjedima.

- **Operator rješenja** $S(i)$ uzima problem $P(i)$ s poznatim aproksimativnim rješenjem $x(i)$ i računa poboljšano rješenje (novi $x(i)$) za taj isti problem.

  $$x_{\text{improved}}(i) = S(i)(b(i), x(i)).$$

  Poboljšanje se dobiva prigušivanjem "komponenti visokih frekvencija" u vektoru pogreške. Što to znači, objasnit ćemo malo kasnije. I taj se operator implementira težinskim usrednjavanjem s najbližim susjedima, a može se interpretirati i kao varijacija Jacobijeve iterativne metode za rješavanje linearnog sustava problema $P(i)$.

Na temelju ovog grubog opisa operatora, bez detalja implementacije, odmah možemo procijeniti njihovu efikasnost. Budući da svaki od ova tri operatora mijenja vrijednosti u pojedinoj točki težinskim sredinama vrijednosti u samoj toj točki i konstantnom broju susjeda, za svaku točku trebamo (najviše) konstantan broj računskih operacija, pa je složenost svakog operatora za n nepoznanica jednaka $O(n)$, tj. linearna u n.

1.2.1. Multigrid V–ciklus

Funkcionalni opis operatora (bez detalja implementacije) dovoljan je za formulaciju osnovnog algoritma koji se naziva **Multigrid V-ciklus** (skraćeno MGV).

```
function MGV( b(i), x(i) )    ... vraca poboljsano rjesenje x(i) za P(i)
  if i = 1    ... samo jedna varijabla
    izracunaj egzaktno rjesenje x(1) problema P(1)
    return ( b(1), x(1) )
  else
    (1) x(i) = S(i)( b(i), x(i) )    ... poboljsaj rjesenje x(i)
    (2) r(i) = T(i) * x(i) - b(i)    ... izracunaj rezidual
    (3) d(i) = In( MGV( 4 * R(r(i)), 0 ))    ... rijesi rekurzivno na grubljoj mrezi
    (4) x(i) = x(i) - d(i)    ... korigiraj rjesenje na finoj mrezi
    (5) x(i) = S(i)( b(i), x(i) )    ... dodatno poboljsaj rjesenje
    return ( b(i), x(i) )
  endif
```

Drugim riječima, algoritam radi sljedeće korake.

(1) Starta s problemom na finoj mreži $(b(i), x(i))$.

(2) Poboljšava rješenje prigušivanjem visokih frekvencija greške

$$x(i) = S(i)(b(i), x(i)).$$

(3) Računa rezidual $r(i)$ aproksimativnog rješenja $x(i)$.

(4) Aproksimira rezidual na sljedećoj grubljoj mreži kao restrikciju $R(r(i))$ ovog reziduala $r(i)$ na finijoj mreži. Taj grublji rezidual će biti desna strana za problem na grubljoj mreži.

(5) Rješava grublji problem rekurzivno, s nulom kao početnom aproksimacijom rješenja

$$MGV(4 * R(r(i)), 0).$$

Faktor 4 dolazi zbog h^2 faktora na desnoj strani Poissonove jednadžbe, koji se, naravno, promijeni za faktor 4, kad s finije mreže prelazimo na 2 puta grublju mrežu.

(6) Preslikava rješenje s grublje mreže na finiju

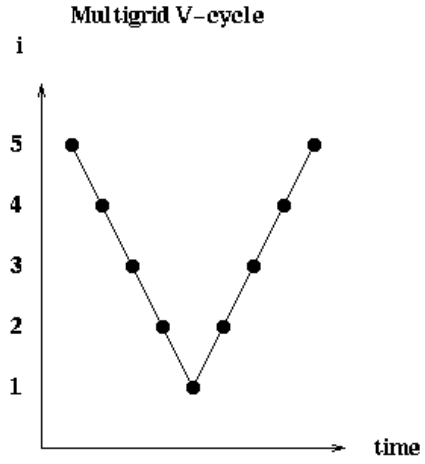
$$d(i) = In(MGV(4 * R(r(i)), 0)).$$

(7) Oduzima korekciju nastalu interpolacijom one izračunate na grubljoj mreži, od rješenja na finijoj mreži

$$x(i) = x(i) - d(i).$$

(8) Još jednom dodatno poboljšava to rješenje $x(i) = S(i)(b(i), x(i))$.

Algoritam se zove V-ciklus, jer ako ga shematski nacrtamo u koordinatnom sustavu s varijablama (broj čvora i, vrijeme), u točkama za rekurzivno zvanje MGV-a, on izgleda kao na slici dolje (početak za $MGV(P(5), x(5))$) u gornjem lijevom kutu. Algoritam zove MGV redom na mrežama 4, 3, do 1, a zatim se vraća na 5.



Kako algoritam ugrubo radi? Neka je $d(i)$ egzaktno rješenje jednadžbe, onda je

$$T(i) \cdot d(i) = r(i) = T(i)x(i) - b(i).$$

Preuređivanjem dobivamo

$$T(i)(x(i) - d(i)) = b(i),$$

pa je $(x(i) - d(i))$ željeno rješenje.

Sada napravimo analizu sekvencijalnog algoritma. U svakoj fazi algoritma proporcionalan je broju nepoznanica (usrednjavanje). Dakle, svaki V ciklus ima cijenu $(2^i - 1)^2 = O(4^i)$ operacija, pa je ukupna količina posla u sekvencijalnom algoritmu

$$\sum_{i=1}^{m} O(4^i) = O(4^m) = O(\text{broj nepoznanica}).$$

Na PRAM računalu sve težinske sredine možemo izračunati u $O(1)$, ako imamo po jedan procesor za svaki čvor mrevze, pa je vrijeme PRAM algoritma jednako broju "točkica" V-ciklusu, tj. $O(m) = O(\log(\text{broja nepoznanica})$.

1.2.2. Puni Multigrid

Puni multigrid (engl. Full Multigrid, skraćeno FMG) koristi MGV samo kao građevni blok.

function FMG(b(m), x(m)) ... vraca tocno rjesenje x(m) za P(m)
  rijesi P(1) egzaktno da dobijes x(1)
  for $i = 2$ to m do
    x(i) = MGV(b(i), In(x(i-1))

Drugim riječima, algoritam radi sljedeće.

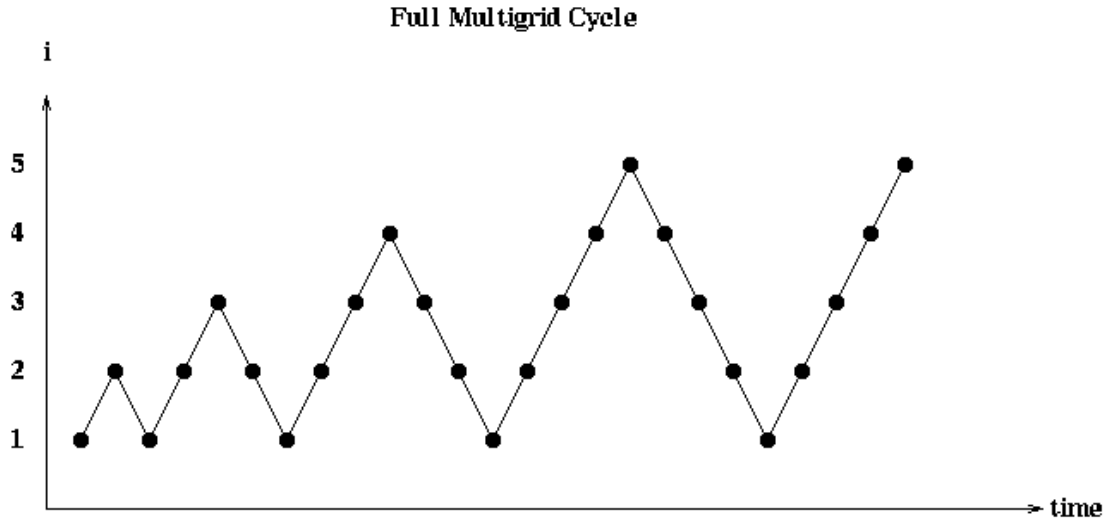
- Rješava problem $P(1)$ egzaktno.
- Rješenje $x(i-1)$ grubljeg problema $P(i-1)$ preslikava kao startnu aproksimaciju $x(i)$ sljedećeg finijeg problema $P(i)$:

$$In(i-1)(b(i-1), x(i-1)).$$

- Rješava finiji problem korištenjem MVG-a s tom startnom aproksimacijom

$$MGV(In(i-1)(b(i-1), x(i-1))).$$

Sada ponovno možemo izračunati složenost algoritma.



Svaki V na prethodnoj slici predstavlja jedan poziv MGV-a u unutarnjoj petlji FMG-a. Na sekvencijalnom računalu, V koji starta na nivou i treba $O(4^i)$ operacija. Zbog toga je ukupno sekvencijalno vrijeme

$$\sum_{i=1}^{m} O(4^i) = O(4^m) = O(\text{broj nepoznanica}).$$

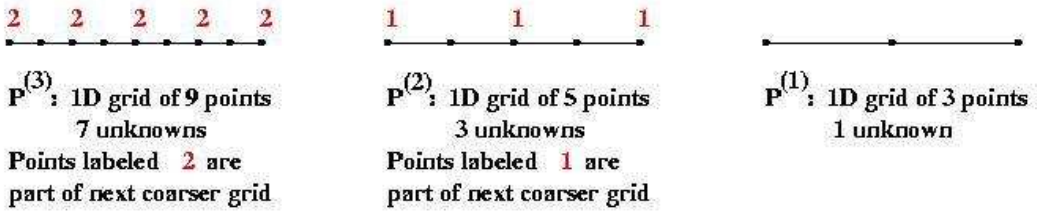
Na PRAM računalu je trajanje suma trajanja svih V ciklusa:

$$\sum_{i=1}^{m} O(i) = O(m^2) = O(\log(\text{broj nepoznanica})^2).$$

Primijetimo da je složenost sekvencijalnog algoritma optimalna, tj. za svaki čvor se koristi konstantan broj operacija. Nasuprot tome, PRAM složenost je kvadrat optimalne (koja se dostiže FFT-om), jer ovdje imamo $O(\log(\text{broj nepoznanica})^2)$ umjesto $O(\log(\text{broj nepoznanica}))$. Nakon detaljnog razmatranja operatora $R(i-1)$, $In(i)$ i $S(i)$ moći ćemo u razmatranje uzeti i cijenu komunikacija.

1.3. Detaljni opis Multigrid metode u 1D

Da bi izlaganje bilo jednostavnije, koristimo 1D problem za objašnjenje detalja algoritma. Tada $P(i)$, kao što je prikazano na slici dolje ima $2^i + 1$ čvorova s $2^i - 1$ (srednjih) nepoznanica.



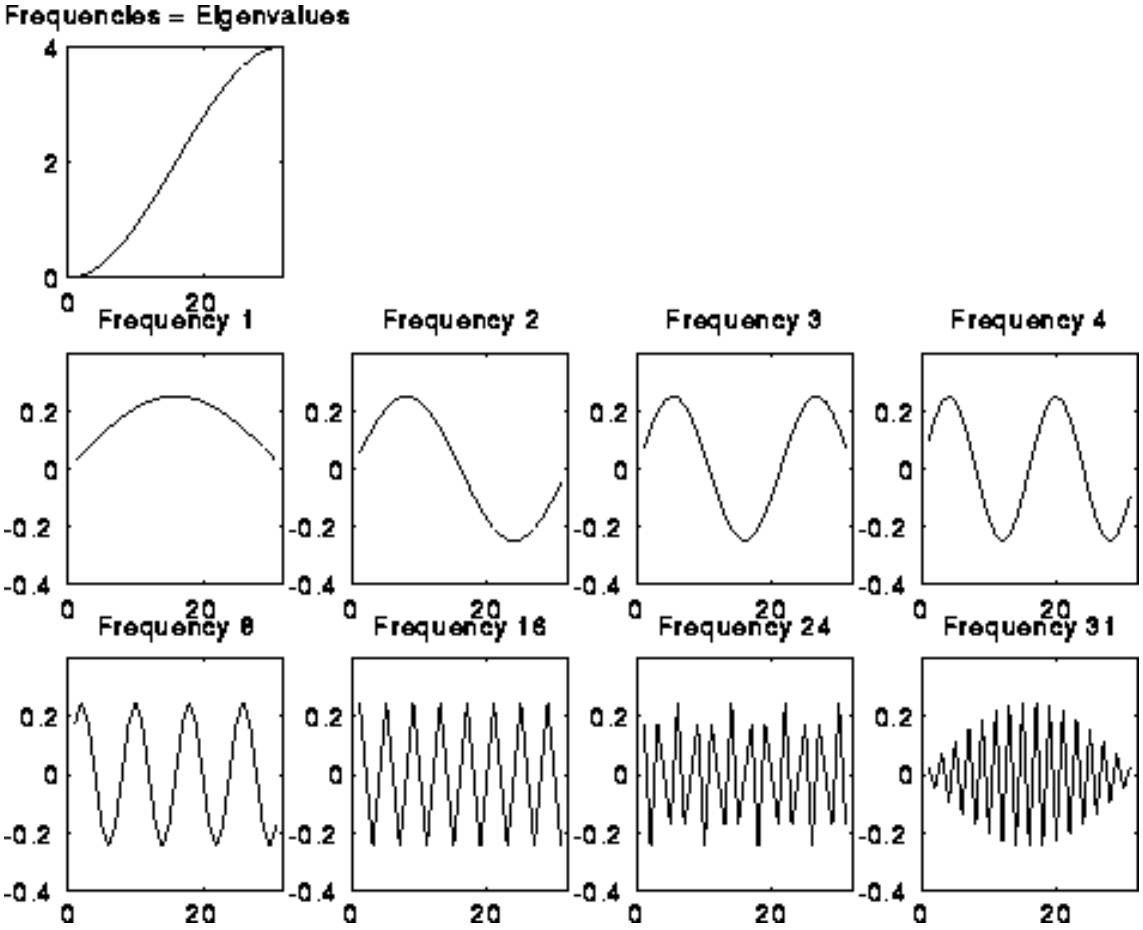
Neka je $T(i)$ matrica koeficijenata problema $P(i)$, a to je skalirana $(2^i - 1) \times (2^i - 1)$ trodijagonalna matrica s 2 na dijagonali i -1 na vandijagonali. Faktor skale 4^{-i} je član $1/h^2$ koji dolazi od toga što $T(i)$ aproksimira drugu derivaciju na mreži s razmakom čvorova $h = 2^{-i}$.

$$T(i) = 4^{-i} \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

Da bismo razumjeli zašto multigrid funkcionira tako dobro pogledajmo vektor rješenja i greške u rješenju kao linearne kombinacije odgovarajućih svojstvenih vektora (koji su ortogonalni, jer je $T(i)$ simetrična), a komponente su im sinusi raznih frekvencija.

Sljedeće slike prikazuju svojstvene vrijednosti i vektore za $i = 5$, tj. za matricu $T(5)$ dimenzija $N \times N$, gdje $N = 2^5 - 1 = 31$.

Najgornji graf predstavlja svojstvene vrijednosti (frekvencije) matrice $4^5 * T(5)$ u rastućem poretku. Sljedeće slike predstavljaju komponente nekih svojstvenih vektora (sinusne krivulje) za pripadne svojstvene vrijednosti (frekvencije). Slike su poredane rastuće po pripadnim svojstvenim vrijednostima. Na početku su slike za najmanje 4 frekvencije, zatim za još nekoliko njih, a posljednja slika odgovara najvišoj frekvenciji (indeksa 31).



Neka je $z(j)$ j-ti svojstveni vektor matrice $T(i)$. Njegove su komponente

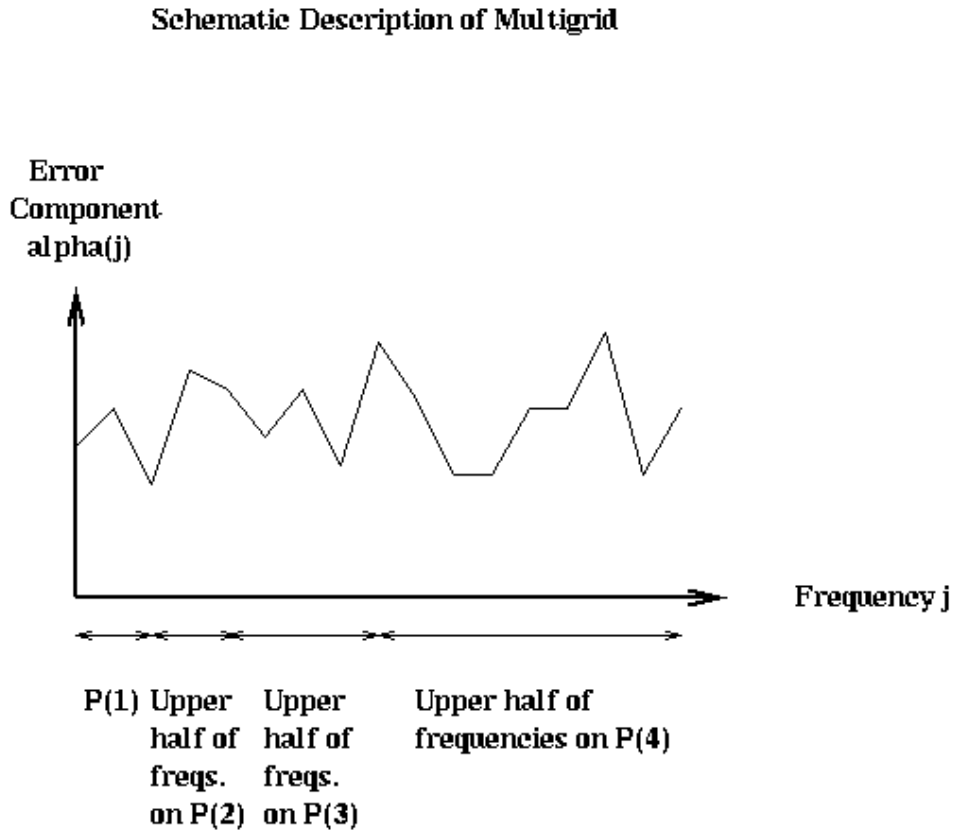
$$z(j,k) = \frac{2}{N+1} \sin \frac{jk\pi}{N+1}.$$

Neka je Z matrica svojstvenih vektora

$$Z = [z(1), z(2), \ldots, z(N)].$$

U sljedećem poglavlju pokazat ćemo kako operator rješenja "ruši" ili "guši" najgornjih pola frekvencija.

Upotrebom ove terminologije, lako se vidi da multigrid metodu možemo opisati i na sljedeći način. Multigrid na najfinijoj mreži $P(m)$ prigušuje najgornjih pola komponenata frekvencije pogreške. To se obavlja korištenjem operatora rješenja $S(i)$, kao što ćemo to još pokazati. U sljedećoj grubljoj mreži multigrid prigušuje polovinu od preostale gornje polovine komponenata frekvencije u greški, sve dok ne dođemo do egzaktnog rješenja za problem $P(1)$. Shematski to izgleda ovako:



1.3.1. Operator rješenja S

U ovom poglavlju konstruirat ćemo operator rješenja i pokazati kako on guši gornju polovinu frekvencija. Zbog jednostavnosti pisanja, nadalje ćemo izostavljati indekse i, tako da umjesto $T(i)$, $x(i)$ i $b(i)$, pišemo samo T, x i b. Pritom ćemo skalom 4^{-i} pomnožiti cijelu jednadžbu, tako da će ona ući u vektor desne strane b.

Pokazat ćemo da je operator rješenja $S(i)$ zapravo "težinska" Jacobijeva iterativna metoda. Standardna Jacobijeva metoda rješenja sustava

$$Tx = b$$

glasi

$$x^{(m+1)} = R_{\text{Jac}} x^m + c_{\text{Jac}},$$

pri čemu je

$$R_{\text{Jac}} = D^{-1}(\widetilde{L} + \widetilde{U}) = \frac{1}{2}\begin{bmatrix} 0 & 1 & & & \\ 1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & 1 \\ & & & 1 & 0 \end{bmatrix} = I - \frac{1}{2}T, \quad c_{\text{Jac}} = D^{-1}b = \frac{1}{2}b.$$

Dakle, standardna Jacobijeva metoda j-tu komponentu približnog rješenja $x^{(i)}$ zamjenjuje sa:

$$\begin{aligned} \text{improved x}^{(i)}(j) &= 0.5(x^{(i)}(j-1) + x^{(i)}(j+1) + 4^i \cdot b(j)) \\ &= x^{(i)}(j) + 0.5(x^{(i)}(j-1) - 2 \cdot x^{(i)}(j) + x^{(i)}(j+1) + 4^i \cdot b(j)). \end{aligned}$$

Težinska Jacobijeva metoda umjesto matrice R_{Jac} uzima težinsku matricu R_w, a umjesto c_{Jac} uzima c_w,

$$R_w = I - \frac{w}{2}T, \quad c_w = \frac{w}{2}b.$$

Time smo dobili težinsku Jacobijevu matodu koja glasi

$$\text{improved x}^{(i)}(j) = x^{(i)}(j) + 0.5w(x^{(i)}(j-1) - 2 \cdot x^{(i)}(j) + x^{(i)}(j+1) + 4^i \cdot b(j)).$$

Prvo primijetimo nekoliko činjenica vezanih uz matrice R_{Jac} i R_w. Ako je svojstvena dekompozicija matrice T bila

$$T = Z\Lambda Z^T,$$

onda zato što su i R_{Jac} i R_w matrični polinomi u T, vrijedi da je njihova svojstvena dekompozicija jednaka

$$R_{\text{Jac}} = Z\left(I - \frac{1}{2}\Lambda\right)Z^T, \quad R_w = Z\left(I - \frac{w}{2}\Lambda\right)Z^T.$$

Tu činjenicu koristit ćemo pri dokazu konvergencije, jer, prisjetimo se, da bi iterativna metoda konvergirala mora biti $\text{spr}(R) < 1$.

Neka je $e^{(m)}$ greška m-te iteracije težinske Jacobijeve metode, tj. neka je

$$e^{(m)} = x^{(m)} - x.$$

Onda imamo

$$\begin{aligned} e^{(m)} &= R_w x^{(m-1)} + c_w - R_w x - c_w = R_w(x^{(m)} - x) = R_w e^{(m-1)} = R_w^m e^{(0)} \\ &= \left(Z\left(I - \frac{w}{2}\Lambda\right)Z^T\right)^m e^{(0)} = Z\left(I - \frac{w}{2}\Lambda\right)^m Z^T e^{(0)}, \end{aligned}$$

pa je

$$Z^T e^{(m)} = \left(I - \frac{w}{2}\Lambda\right)^m Z^T e^{(0)},$$

odnosno j-ta komponenta tog vektora je

$$(Z^T e^{(m)})(j) = \left(I - \frac{w}{2}\Lambda\right)^m_{j,j} (Z^T e^{(0)})(j),$$

j-tu komponentu vektora $(Z^T e^{(m)})(j)$ zovemo j-ta komponenta frekvencije greške $e^{(m)}$, budući da je

$$e^{(m)} = Z(Z^T e^{(m)}),$$

pa je to težinska suma stupaca matrice Z gdje je težina $(Z^T e^{(m)})(j)$. Ako znamo da su komponente Z sinusoide raznih frekvencija, onda će svojstvene vrijednosti matrice R_w

$$\lambda_j(R_w) = 1 - \frac{w}{2}\lambda_j$$

određivati kojom će se brzinom svaka od komponenti prigušivati.

Ako uzmemo $w = 2/3$, onda vrijedi

$$|\lambda_j(R_w)| = |1 - \frac{1}{3}\lambda_j|,$$

gdje je

$$\lambda_j = 2\left(1 - \cos\frac{\pi j}{N+1}\right),$$

pa za $j > N/2$ imamo

$$2 < \lambda_j < 4,$$

pa je za te j

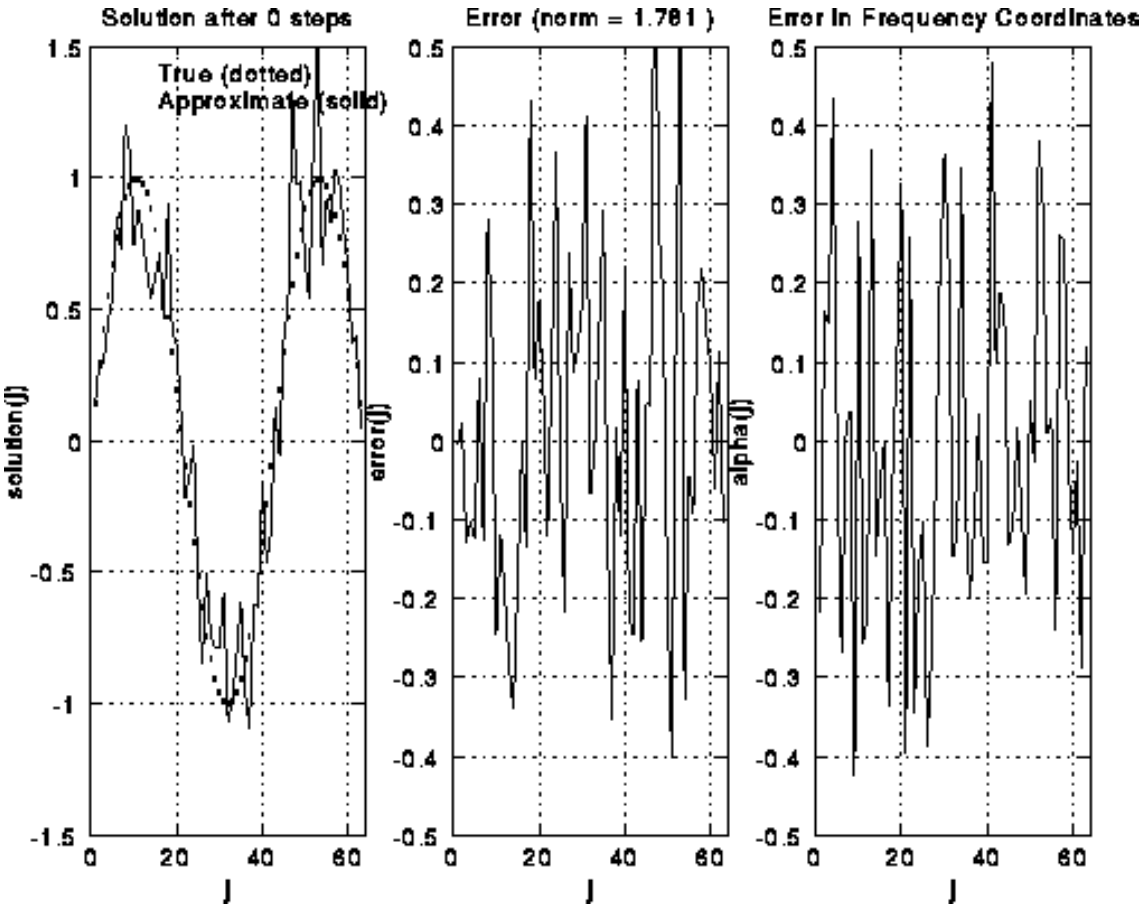
$$|\lambda_j(R_w)| < \frac{1}{3}.$$

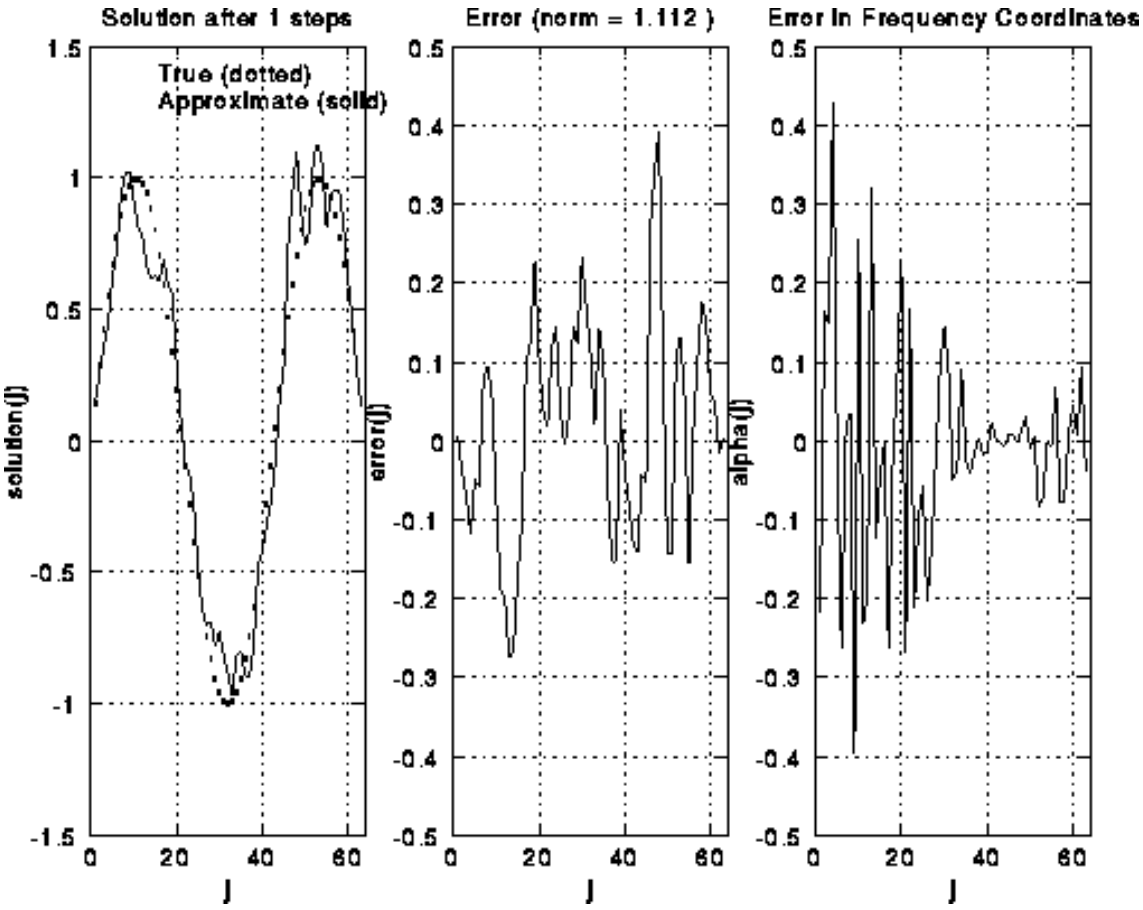
To znači da se u težinskoj Jacobijevoj metodi greške gornje polovine komponenata množe s 1/3 ili manje u svakoj iteraciji, bez obzira na N. Taj w je, dakle dobar izbor, pa težinska Jacobijeva metoda za $S(i)$ glasi

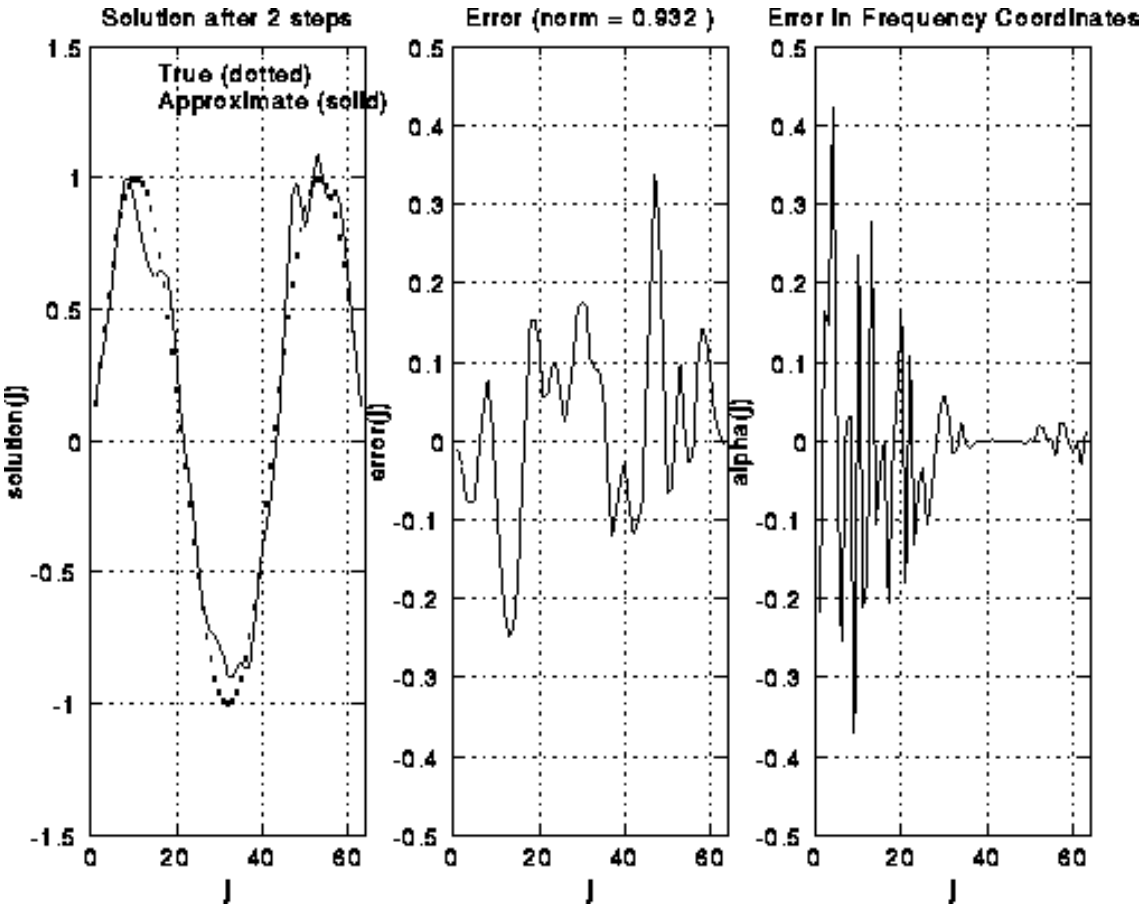
$$\text{improved x}^{(i)}(j) = \frac{1}{3}(x^{(i)}(j-1) + x^{(i)}(j) + x^{(i)}(j+1) + 4^i b(j)).$$

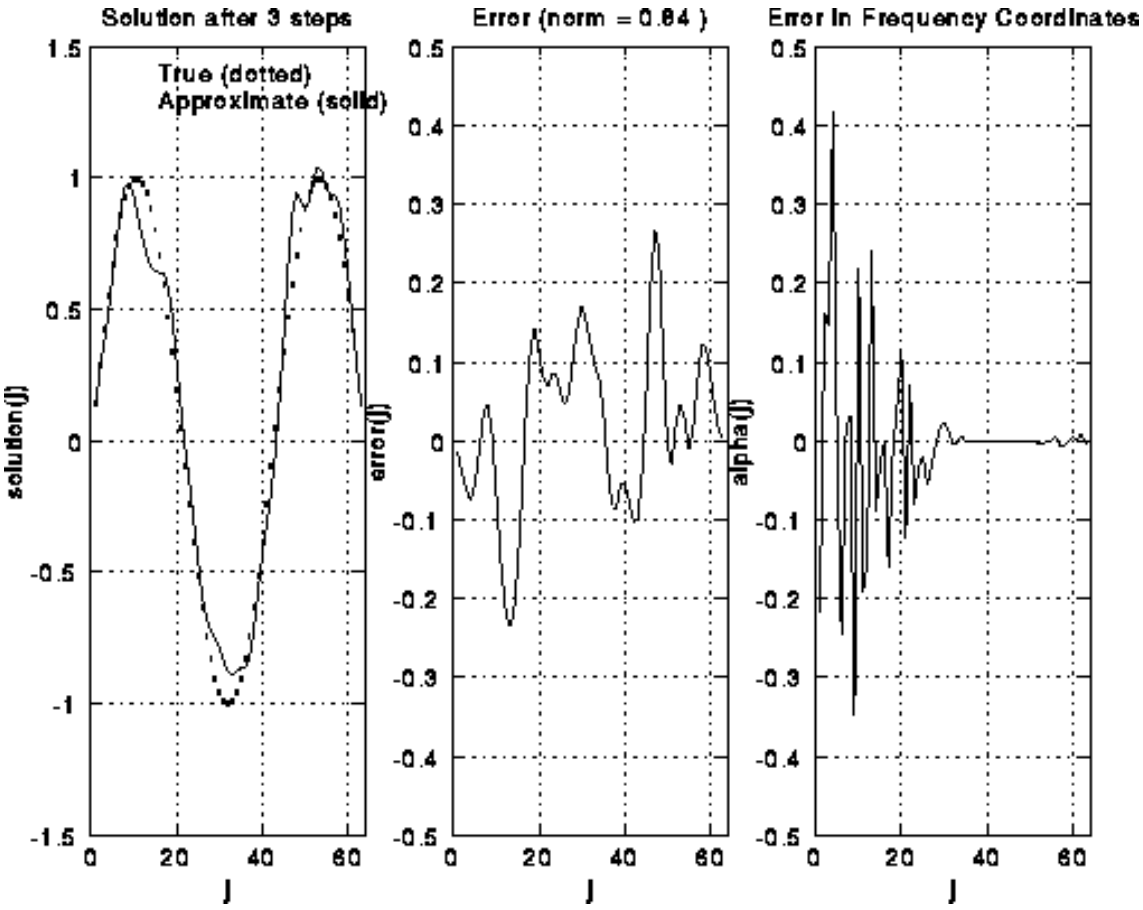
Evo ilustracija za $S(i)$, ako je $i = 6$, a $N = 2^6 - 1 = 63$ (dakle imamo 63 nepoznanice). Imamo 6 redova slika — u prvom je rješenje i greška početnog rješenja, a preostalih 5 pokazuju rješenja i greške nakon uzastopnih primjena $S(i)$. Pravo je rješenje sinusoida nacrtana točkasto u najljevijoj slici u svakom retku. Aproksimativno je rješenje prikazano na istoj slici, ali punom linijom. Srednja slika u retku prikazuje grešku i njenu normu (u naslovu slike). Najdesnija slika pokazuje komponente frekvencija greške $(Z^T e^{(m)})$. Odmah se vidi da nakom primjene $S(i)$, desna

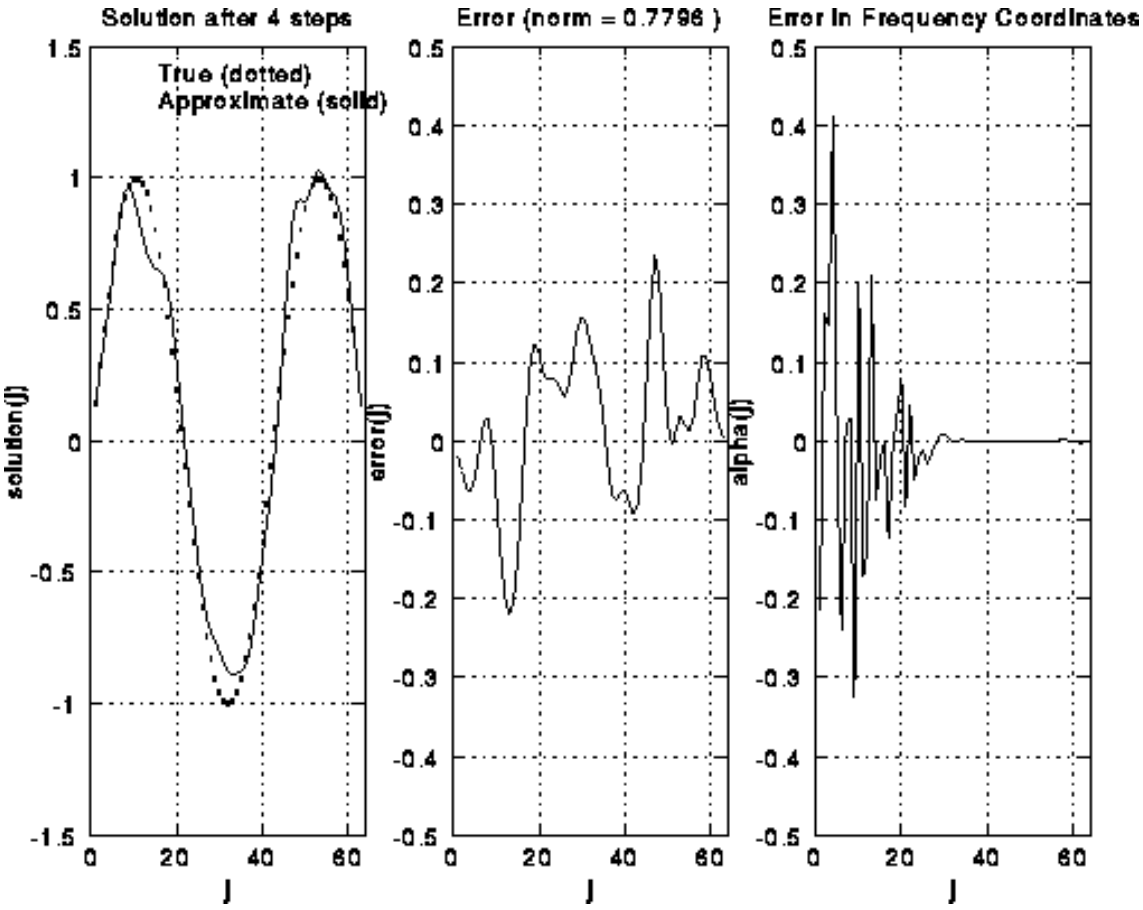
polovina frekvencija se priguši. To također omogućava da aproksimativna rješenja postanu glađa jer greške u nižim frekvencijama izgledaju glađe nego u višim. Na početku norma vektora rapidno opada s 1.78 na 1.11, ali poslje postupno opada, jer treba prigušiti malo više grešaka u visokim frekvencijama. Zbog toga ima smisla upotrijebiti samo 1 do 2 iteracije za $S(i)$ u određenom vremenskom trenutku.

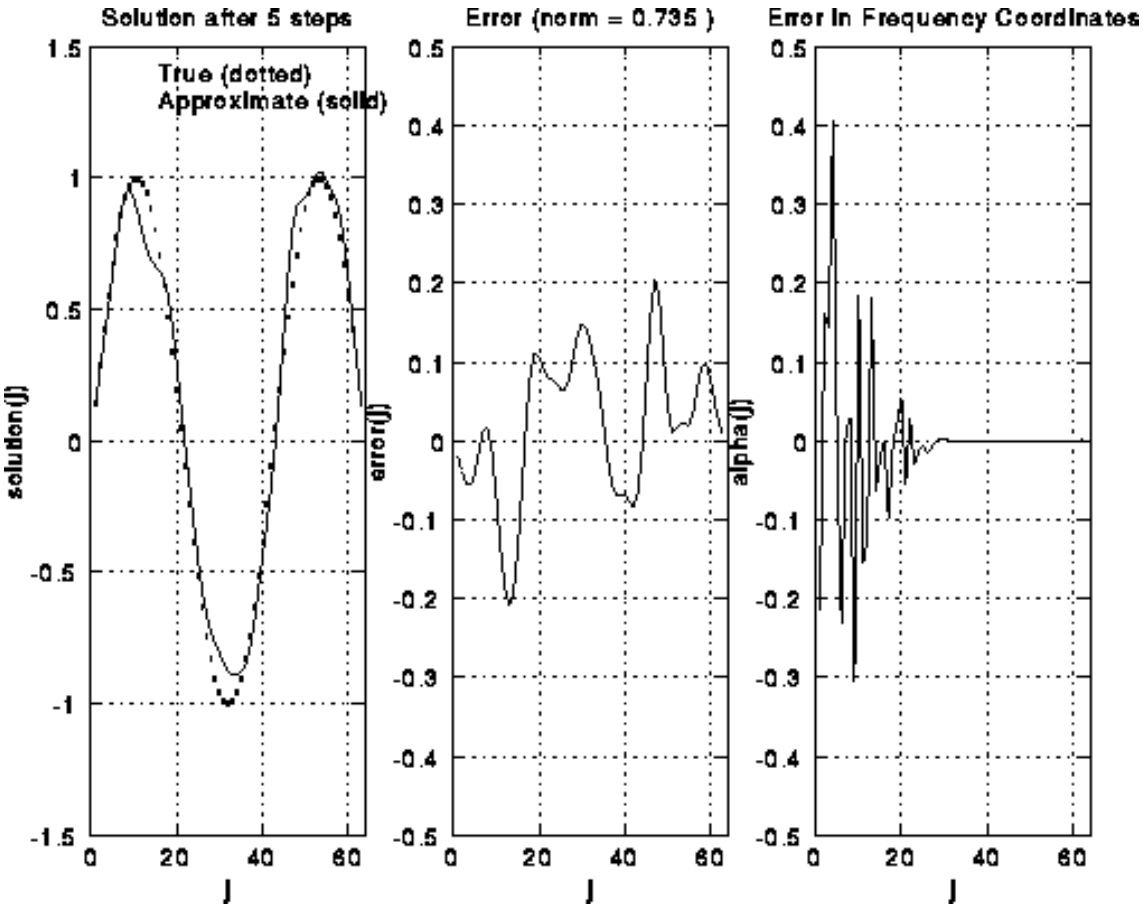


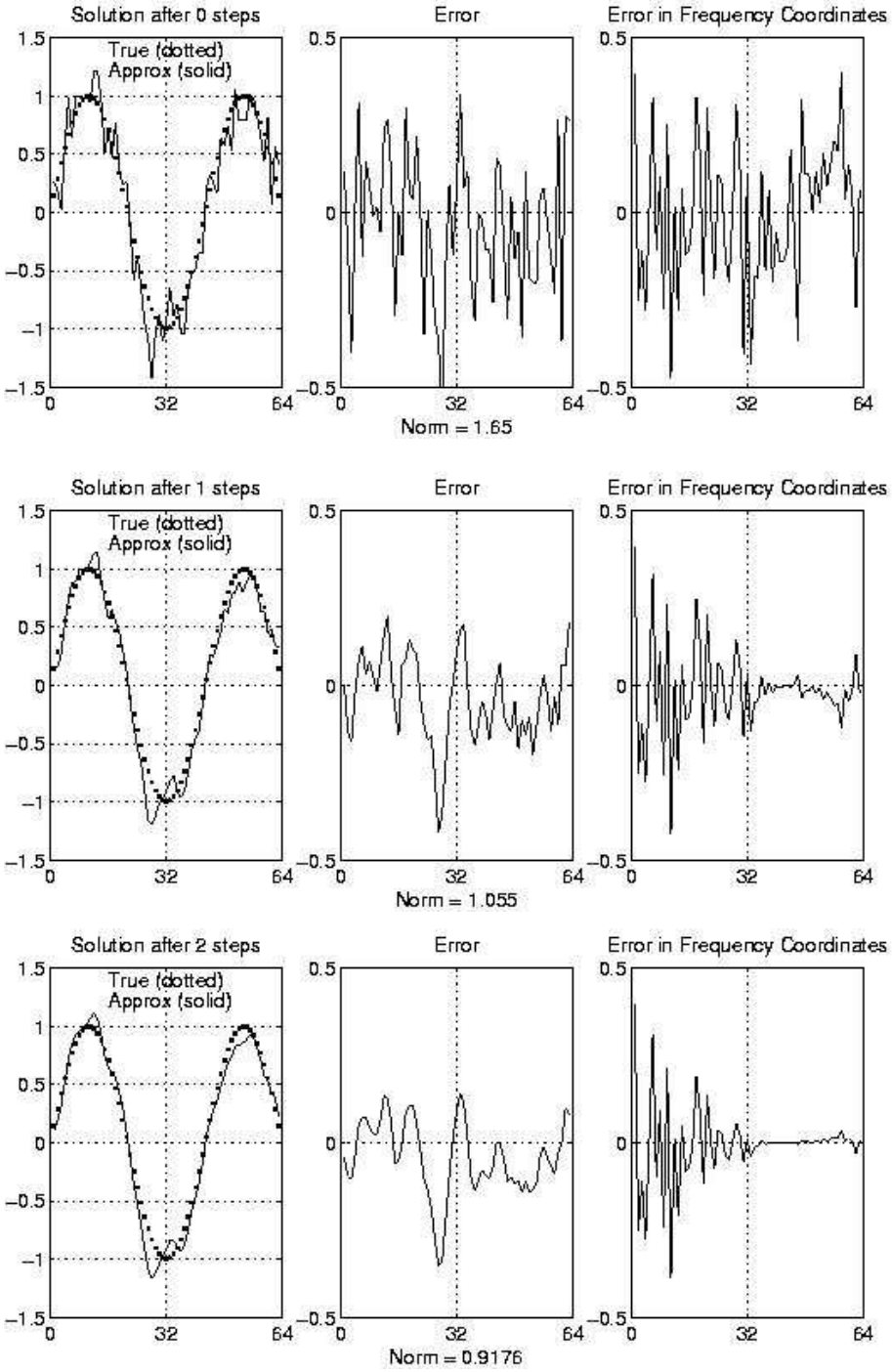

Solution after 1 steps
True (dotted)
Approximate (solid)
solution(j)
J
Error (norm = 1.112)
error(j)
J
Error in Frequency Coordinates
alpha(j)
J

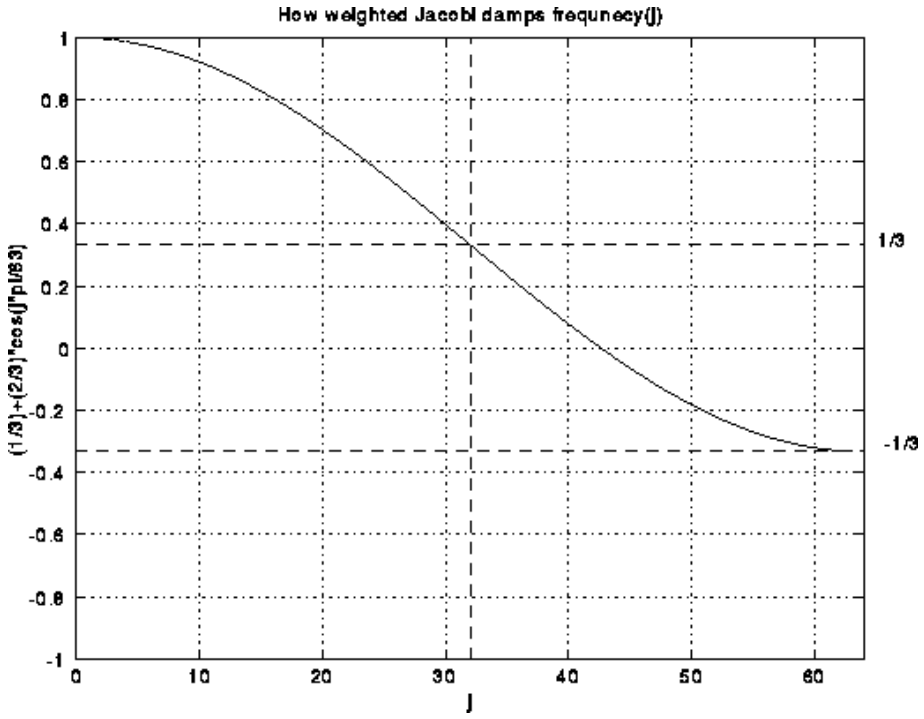

Solution after 2 steps
True (dotted)
Approximate (solid)
solution(j)
j
Error (norm = 0.932)
error(j)
j
Error in Frequency Coordinates
alpha(j)
j


Solution after 3 steps
True (dotted)
Approximate (solid)
solution(j)
j
Error (norm = 0.64)
error(j)
j
Error in Frequency Coordinates
alpha(j)
j


Solution after 4 steps
True (dotted)
Approximate (solid)
solution(j)
j
Error (norm = 0.7796)
error(j)
j
Error in Frequency Coordinates
alpha(j)
j


Solution after 5 steps
True (dotted)
Approximate (solid)
solution(j)
J
Error (norm = 0.735)
error(j)
J
Error in Frequency Coordinates
alpha(j)
J


Solution after 0 steps
True (dotted)
Approx (solid)
Error
Error in Frequency Coordinates
Norm = 1.65
Solution after 1 steps
True (dotted)
Approx (solid)
Error
Error in Frequency Coordinates
Norm = 1.055
Solution after 2 steps
True (dotted)
Approx (solid)
Error
Error in Frequency Coordinates
Norm = 0.9176



U 2D slučaju, djelovanjem operatora $S(i)$ svaka točka bit će zamijenjena sa sličnom težinskom sumom, ali sada od (najviše) 4 susjeda.

Ako operator $S(i)$ za težinsku Jacobijevu metodu u 1D slučaju napišemo u obliku

$$\text{improved } \mathrm{x}_j^{(i)} = (1-w)\cdot x_j^{(i)} + \frac{w}{2}\Big(x_{j-1}^{(i)} + x_{j+1}^{(i)} + b_j^{(i)}\Big),$$

onda operator $S(i)$ u 2D slučaju ima oblik

$$\text{improved } \mathrm{x}_{j,k}^{(i)} = (1-w)\cdot x_{j,k}^{(i)} + \frac{w}{4}\Big(x_{j-1.k}^{(i)} + x_{j+1,k}^{(i)} + x_{j.k-1}^{(i)} + x_{j,k+1}^{(i)} + b_{j,k}^{(i)}\Big).$$

U oba slučaja koristi se težina $w = 2/3$.

1.3.2. Operator restrikcije R

Sada promatrajmo operator restrikcije $R(i)$, koji uzima desnu stranu $b(i)$ problema $P(i)$, i aproksimativno rješenje $x(i)$, i preslikava ga na problem $P(i-1)$ s desnom stranom $b(i-1)$ i aproksimativnim rješenjem $x(i-1)$.

Neka je $r(i)$ rezidual aproksimativnog rješenja $x(i)$:

$$r(i) = T(i)x(i) - b(i).$$

Ako je $x(i)$ egzaktno rješenje, $r(i)$ je 0. Ako rješimo jednadžbu

$$T(i)d(i) = r(i)$$

egzaktno za korekciju $d(i)$, tada je $x(i) - d(i)$ rješenje koje tražimo, jer je

$$T(i)(x(i) - d(i)) = T(i)x(i) - T(i)d(i) = (r(i) + b(i)) - r(i) = b(i).$$

Dakle, ako $R(i)$ primijenimo na $(b(i), x(i))$ dobijemo $P(i-1)$ moramo

- izračunati rezidual $r(i)$,
- restringirati ga na sljedeću grublju mrežu da dobijemo $b(i-1) = r(i-1)$,
- staviti početnu pretpostavku rješenja $x(i-1) = 0$.

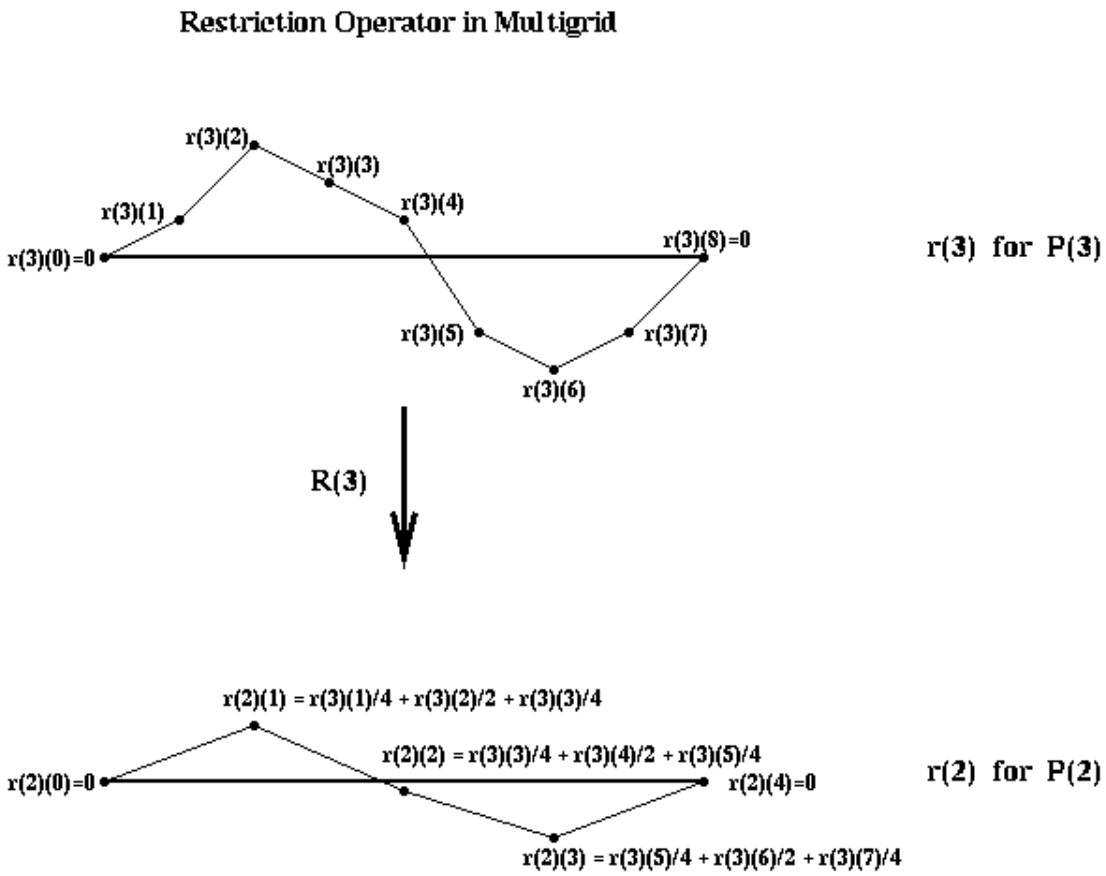
Dakle, $P(i-1)$ ćemo dobiti rješavanjem problema

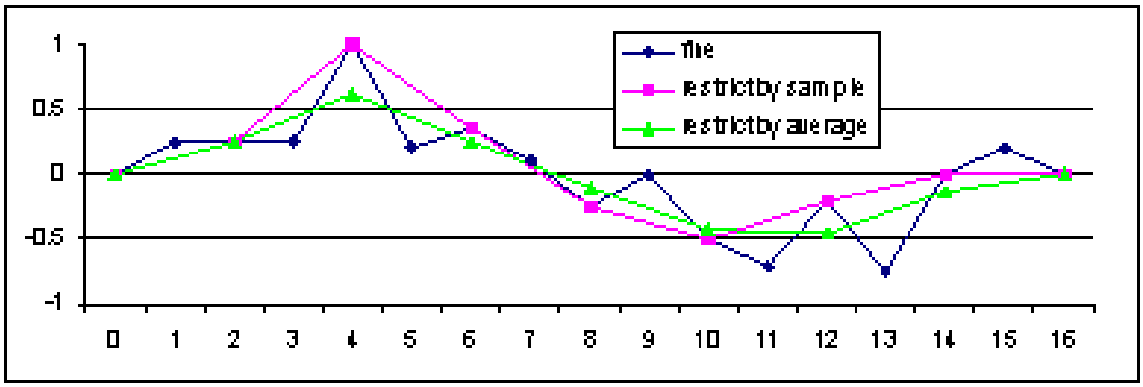
$$T(i-1)d(i-1) = r(i-1),$$

za korekciju $d(i-1)$, startamo li od početne pretpostavke $x(i-1) = 0$ (0 je vektor!).

Restrikcija će se dobiti usrednjavanjem s najbližim susjedima $r(i)$ na finoj mreži (s 2^i-1 nepoznanica) da bi se dobila aproksimacija na grubljoj mreži (a $2^{i-1}-1$ nepoznanica). Vrijednost u grubljoj mreži je 0.5 puta vrijednost u odgovarajućoj točki fine mreže + 0.25 puta vrijednost u 2 susjeda fine mreže. Matrično, to izgleda ovako:

$$b(i-1) = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & & & & \\ & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & & & \\ & & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & & \\ & & & \ddots & \ddots & \ddots & & \end{bmatrix} b(i)$$





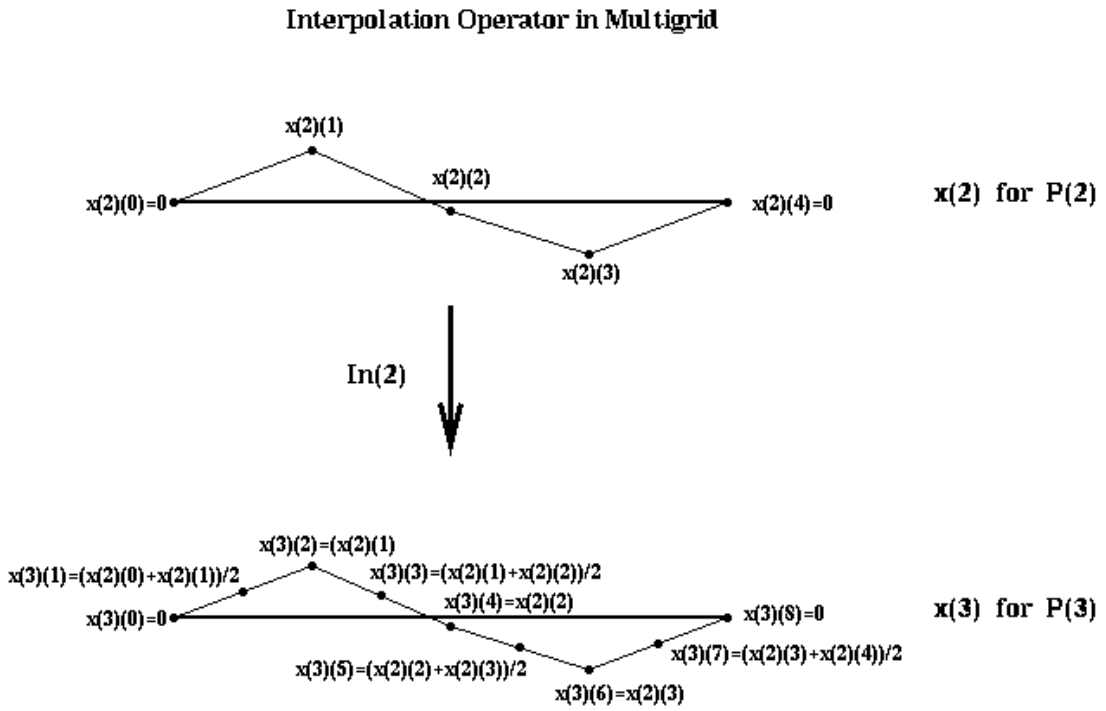
U 2D slučaju, operator $R(i)$ zahtijeva usrednjavanje s (najviše) 8 najbližih susjeda (u N, S, E, W, NW, SW, SE i NE smjerovima prema kompasu). Princip usrednjavanja je isti, samo se primijenjuje u oba smjera.

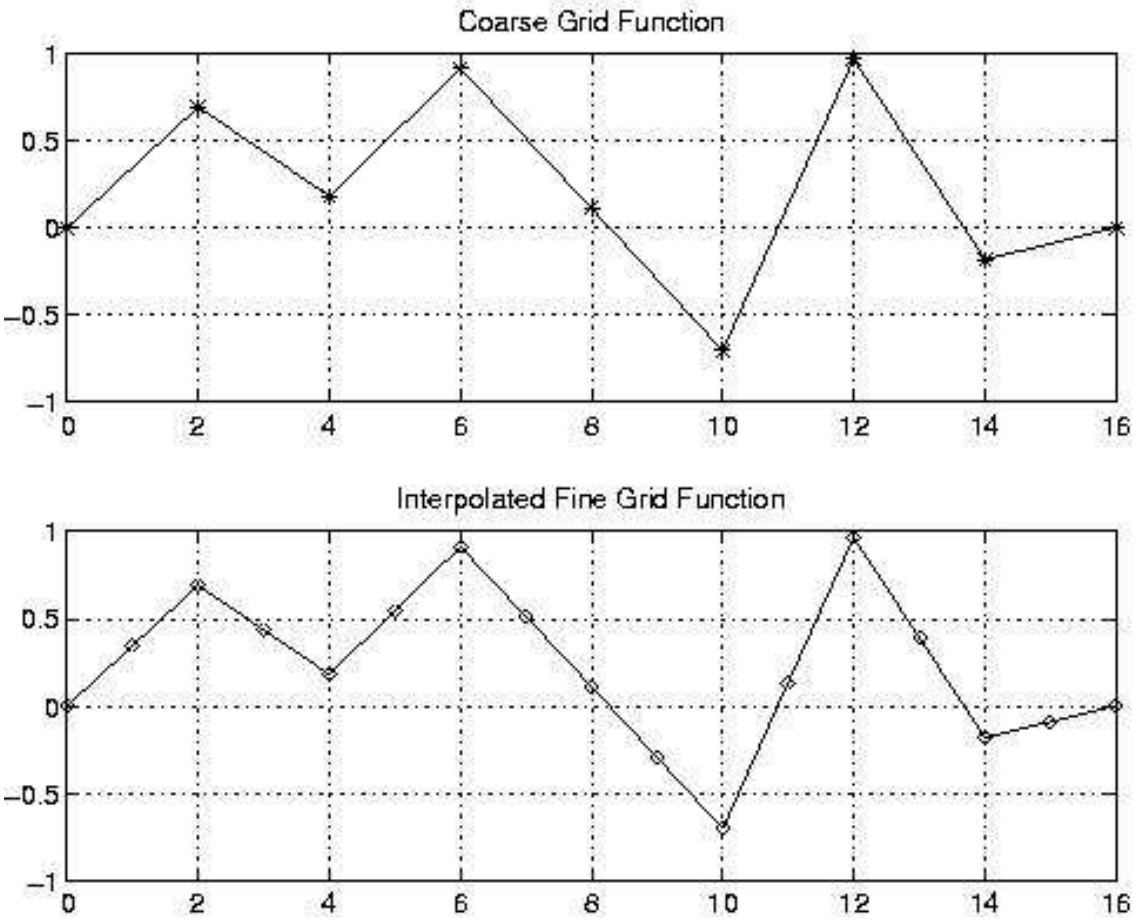
Vrijednost u (unutrašnjoj) točki grublje mreže dobiva se tako da se uzme: 1/4 vrijednosti u toj točki na finijoj mreži, plus 1/8 puta zbroj vrijednosti u susjed-

ima lijevo, desno, gore i dolje (W, E, N, S), plus 1/16 puta zbroj vrijednosti u dijagonalnim susjedima (NW, NE, SE, SW).

1.3.3. Operator interpolacije In

Na kraju, operator interpolacije $In(i-1)$ uzima približno rješenje $d(i-1)$ na grubljoj mreži i preslikava ga u aproksimaciju rješenja $d(i)$ na finijoj mreži. Rješenje $d(i-1)$ se interpolira na finojoj mreži na sljedeći način.





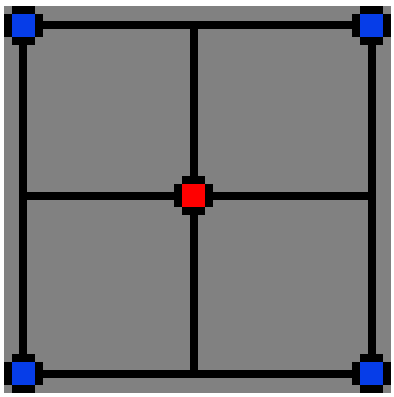
Matematički, interpolaciju rješenja možemo napisati kao

$$d(i) = In(i-1)(d(i-1)) = \begin{bmatrix} \frac{1}{2} & & & \\ 1 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ & 1 & \ddots & \\ & \frac{1}{2} & \ddots & \frac{1}{2} \\ & & \ddots & 1 \\ & & & \frac{1}{2} \end{bmatrix} d(i-1)$$

U 2D slučaju, interpolacija zahtijeva usrednjavanje s (najviše) 4 najbliža susjeda (u NW, SW, SE i NE smjerovima prema kompasu).

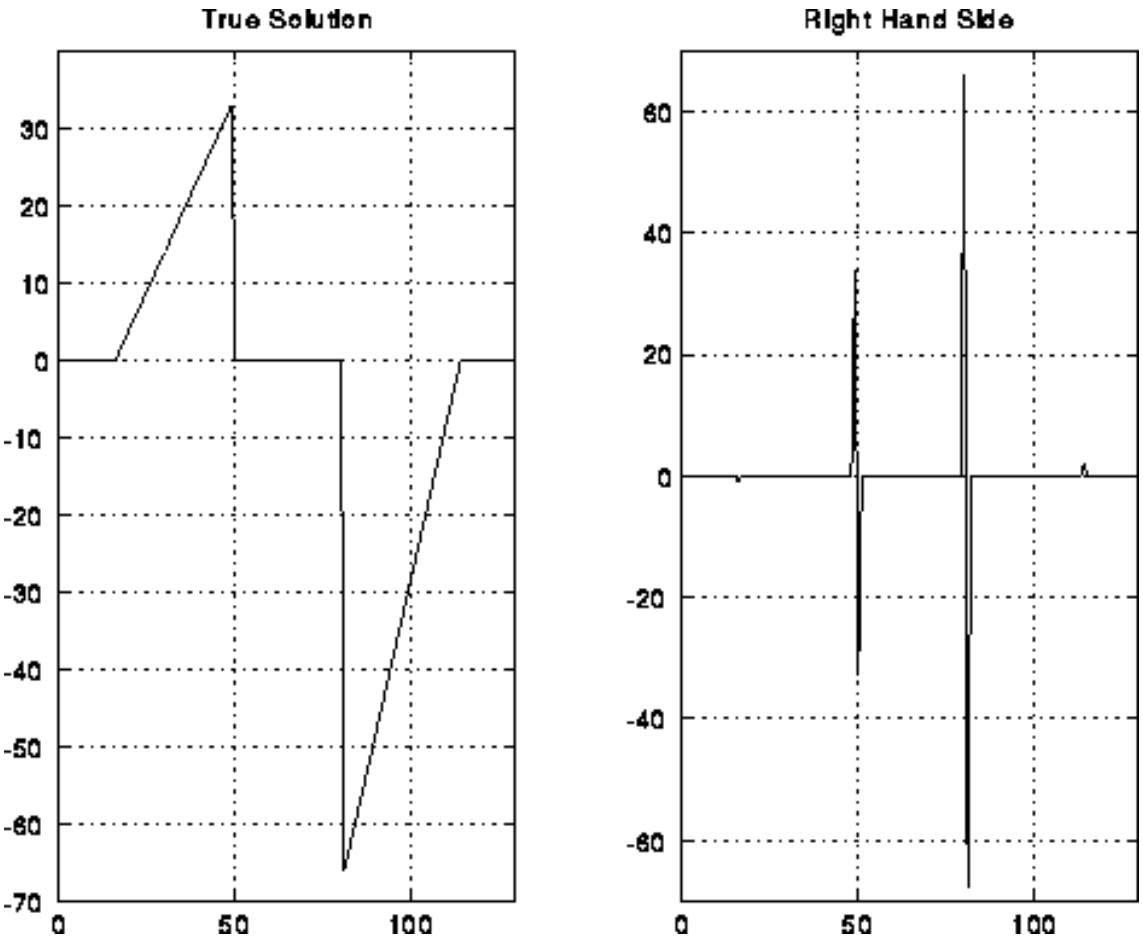
Preciznije, ako je točka finije mreže ujedno i točka grublje mreže, uzimamo istu vrijednost ("usrednjavanje" preko jedne točke). Ako točka finije mreže ima dva najbliža susjeda iz grublje mreže (gore i dolje, ili lijevo i desno), uzimamo srednju vrijednost tih susjeda (usrednjavanje preko dvije točke s faktorom 1/2, kao u 1D slučaju).

Konačno, ako točka finije mreže ima četiri najbliža susjeda iz grublje mreže (dijagonalo raspoređenih, tako da je točka u središtu kvadrata, kao na sljedećoj slici), uzimamo srednju vrijednost tih susjeda (usrednjavanje preko četiri točke s faktorom 1/4).



1.4. Primjeri konvergencije Multigrid metode

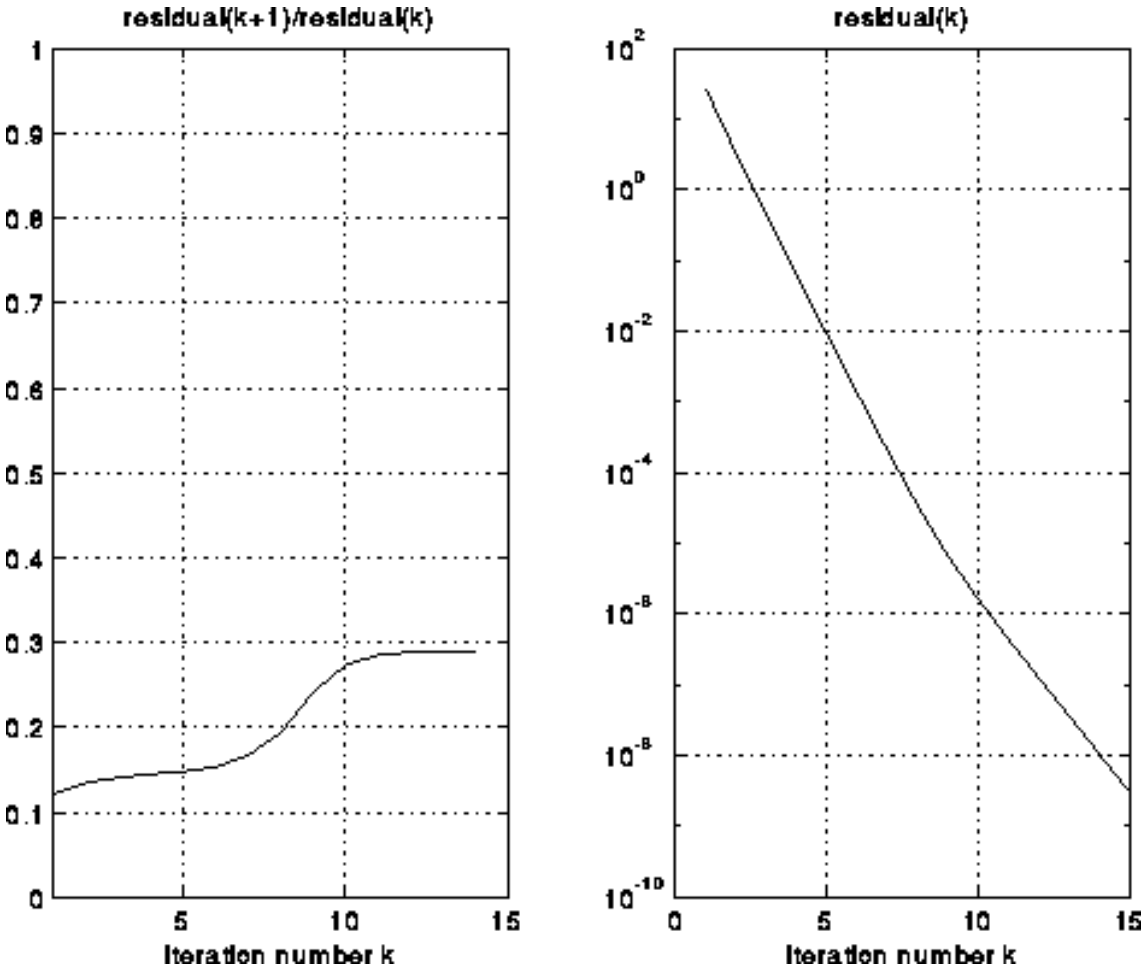
Sada pokažimo kako radi Multigrid na 1D problemu, a zatim i na 2D problemu. Sljedeći graf predstavlja desnu stranu i pravo rješenje 1D problema sa $127 = 2^7 - 1$ nepoznanica.



Sljedeći graf daje pregled konvergencije Multigride i to na dva načina. Graf na desnoj strani predstavlja rezidual

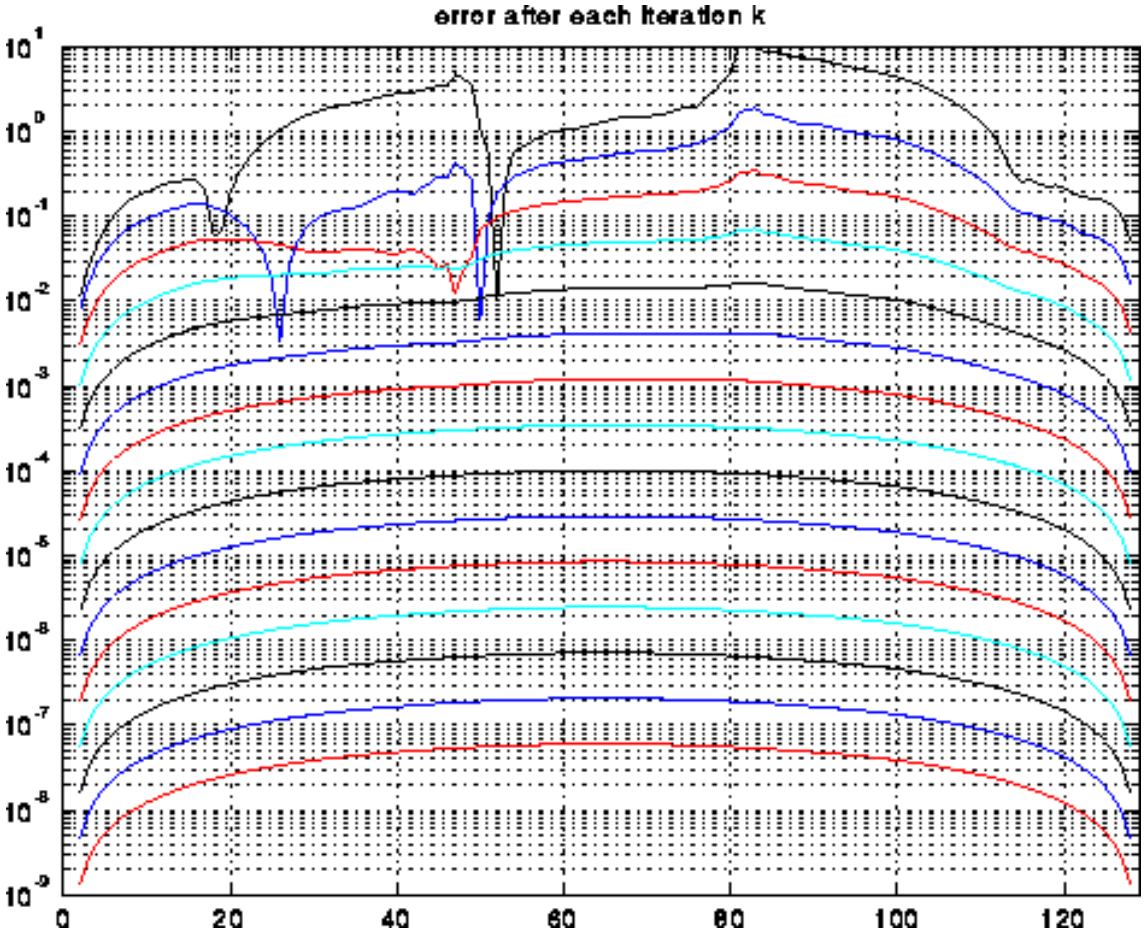
$$residual(k) = \|T(i)x(i) - b(i)\|$$

nakon k iteracija punog Multigrida (FMG). Graf na lijevoj strani predstavlja omjer normi susjednih reziduala, i pokazuje kako je rezidual opao za faktor koji je ograđen od 1 u svakom koraku (za manje glatka rješenja, rezidual će opadati s faktorom 0.5 u svakom koraku.)

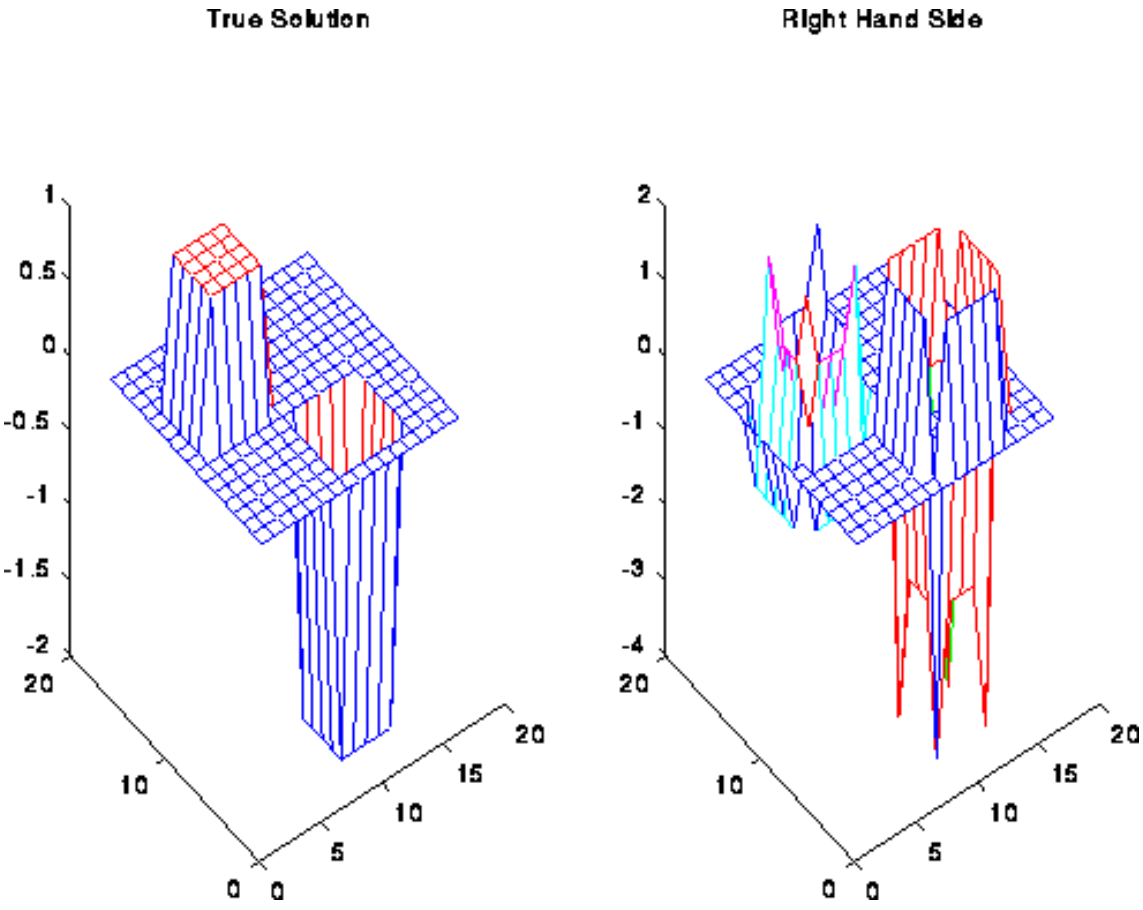


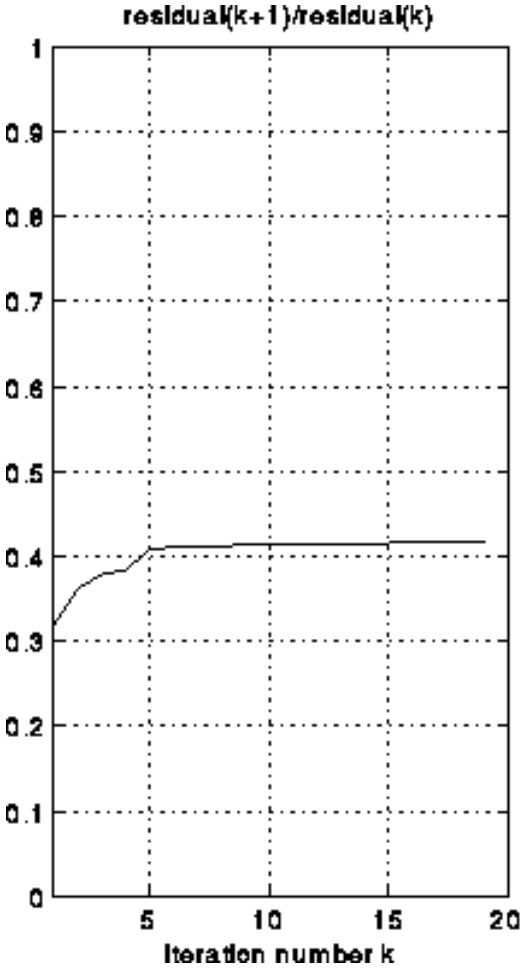
Konačno, sljedeći graf pokazuje kako se ponaša greška za sve iteracije u svakom čvoru mreže. Susjedne iteracije označene su različitim bojama

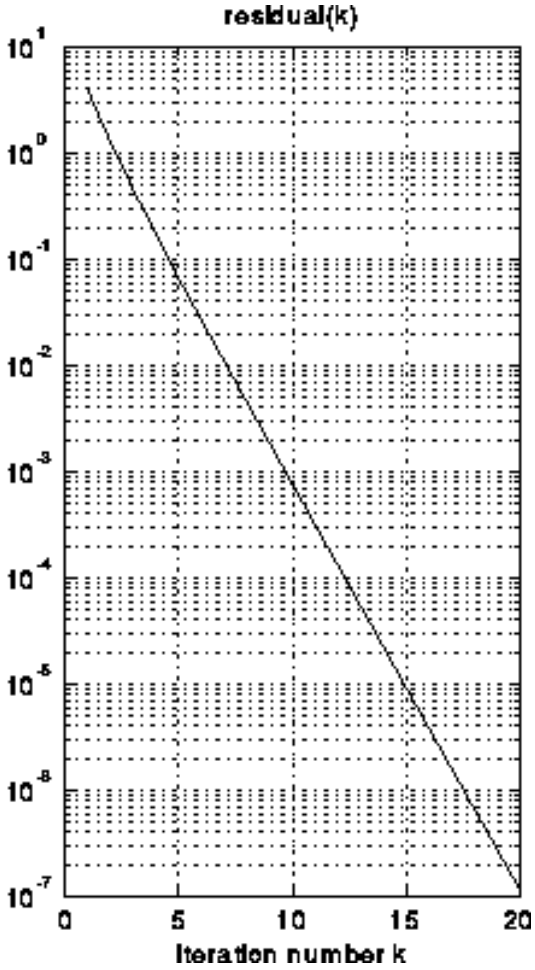
Možemo vidjeti da se greške mijenjaju za konstantni faktor za taj graf u polulogaritamskom mjerilu, što znači da opadaju za konstantni faktor. Također vidimo da greške postaju glađe što dulje iteriramo.



Sljedeće je mala 2D ilustracija. Prva dva grafa su pravo rješenje i desna strana, a sljedeća dva predstavljaju reziduale kao i na 1D crtežu.


True Solution
Right Hand Side
1
0.5
0
-0.5
-1
-1.5
-2
2
1
0
-1
-2
-3
-4
20
10
0
0
5
10
15
20





1.5. Paralelizacija Multigrida

Sada ćemo napraviti pažljiviju analizu paralelne složenosti 2D Multigrid algoritma, koja uključuje i cijenu komuniciranja među procesorima. Gdje će se pojaviti potreba za komunikacijom?

Kao što smo već opisali, kod 2D multigrida nova vrijednost u svakom čvoru mreže ovisi o najviše do 8 susjeda tog čvora — to su susjedi u smjerovima N, E, S, W, NW, SW, SE i NE, prema stranama svijeta (kompasu) obzirom na taj čvor. Tih susjeda može biti i manje, ako smo na rubu mreže. Susjedni čvorovi mogu pripadati različitim procesorima, što rezultira komunikacijom i tako ćemo odrediti njenu cijenu.

Pretpostavimo, kao i dosad, da je $n = 2^m + 1$ i da imamo $n \times n$ mrežu čvorova s podacima u dvije dimenzije. Prirodno je uzeti 2D polje procesora s prirodnim rasporedom podataka po procesorima.

Dakle, neka je broj procesora $p = s^2$ potpuni kvadrat. Podaci su raspoređeni na $s \times s$ mreži procesora, s tim da svaki procesor ima svoju odgovarajuću podmrežu cijele mreže čvorova s podacima, dimenzija

$$\frac{n-1}{s} \times \frac{n-1}{s}.$$

Odmah se vidi da je, radi jednostavnosti, zgodno pretpostaviti da je i s potencija od 2. Uzmimo da je

$$s = 2^k, \quad p = 4^k = 2^{2k},$$

s tim da je logično pretpostaviti da je $k \leq m$, odnosno $p \leq n^2$, tako da imamo barem po jedan čvor cijele mreže po procesoru. Tada svaki procesor ima podmrežu dimenzija

$$2^{m-k} \times 2^{m-k}.$$

Striktno govoreći, još treba po jednu od paralelnih stranica kvadrata (na kojima su vrijednosti rješenja zadane), pridružiti rubnim procesorima na tom rubu — recimo, gornju i lijevu stranicu dodjeljujemo pripadnim procesorima.

Takva situacija je ilustrirana na sljedećoj slici, za $m = 5$ i $k = 2$. Imamo mrežu od 33×33 točaka raspodjeljenu na $4 \times 4 = 16$ procesora.

Ružičasta isprekidana crta omeđuje one točke mreže koje pripadaju odgovarajućem procesoru.

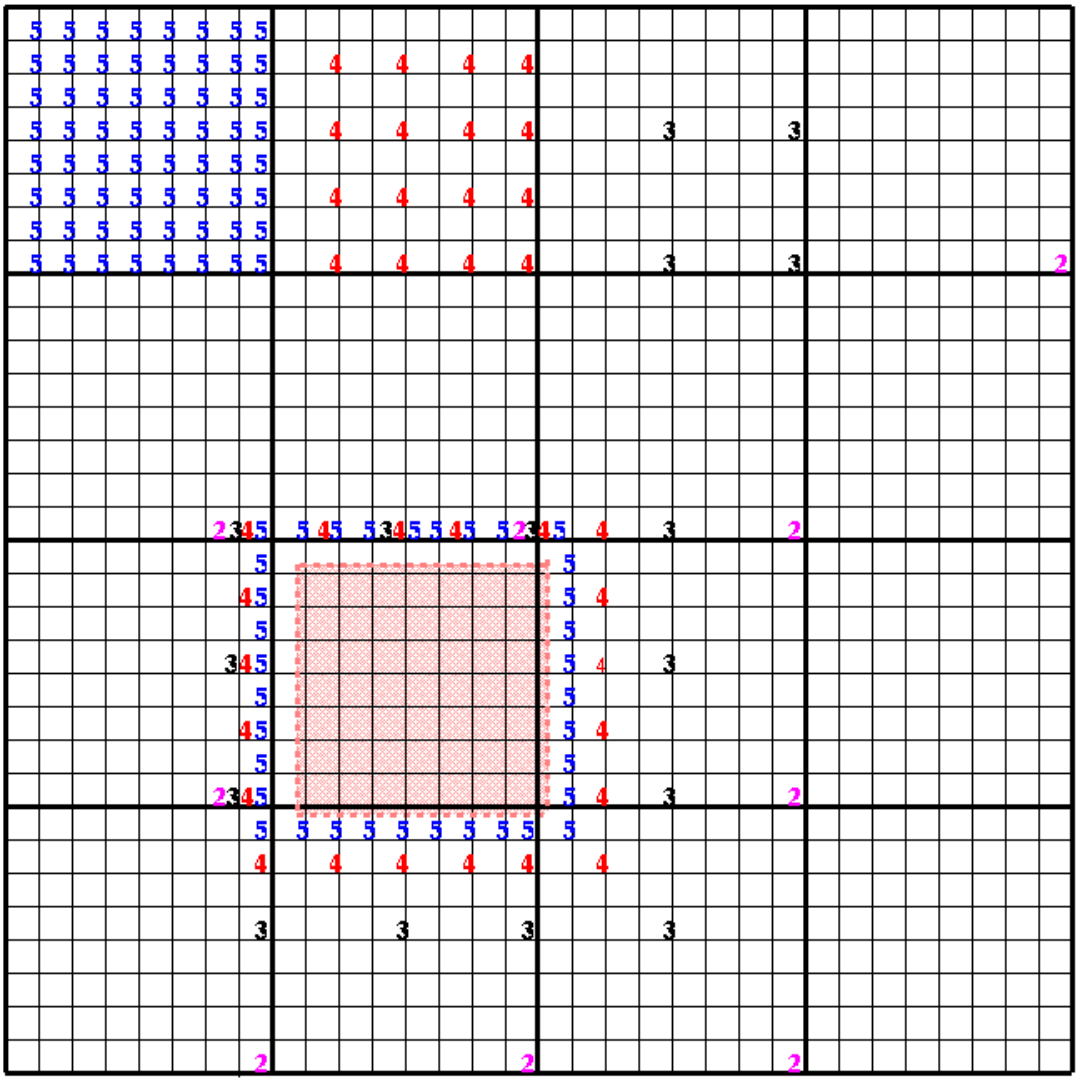
Osim toga, u najgornjem redu procesora, čvorovi mreže posebno su označeni brojem (i bojom) potproblema kojem pripadaju, tako da broj i označava pripadnost problemu $P^{(i)}$ (i svim problemima većeg indeksa).

Uočite da imamo točno po jednu točku označenu 2 po svakom procesoru. Nadalje, jedina unutrašnja (ne-rubna) točka u $P^{(1)}$ je ona u centru kvadrata i pripada procesoru iznad "ružičastog".

U općem slučaju, čim je $k > 0$, odnosno $p > 1$ (što znači $p \geq 4$), postoje problemi $P^{(i)}$ s dovoljno malim brojem čvorova (manjim od broja procesora), tako da poneki procesori uopće ne sadrže pripadne čvorove tog problema (prazni su). To su problemi $P^{(i)}$ za $i < k$, a ima ih

$$k - 1 = \frac{1}{2} \log_2 p - 1.$$

O tome treba voditi računa kad analiziramo cijenu komunikacije, što znači da ćemo imati dva slučaja: $i \geq k$ i $i < k$.



Communication pattern for Multigrid on 33 by 33 mesh with 4 by 4 processor grid
In top processor row, grid points labeled m are updated in problem P(m) of multigrid
Pink processor owns grid points inside pink box
In lower half of graph, grid points labeled m need to be communicated to pink processor in problem P(m) of multigrid

Pogledajmo sad kad imamo komunikaciju među procesorima u pojedinim fazama multigrid algoritma — prvo za jedan V-ciklus, a zatim za puni multigrid ciklus (FMG).

Uzmimo za primjer "ružičasti" procesor s prethodne slike. On treba određene podatke za čvorove od svojih susjeda da bi mogao izvršiti algoritam. Pripadni čvorovi označeni su na isti način kao i prije — indeksom problema i bojom.

Na primjer, da bi izračunao nove vrijednosti rješenja za svoje (plave) točke u najfinijem problemu $P^{(5)}$, "ružičasti" procesor treba:

- vrijednosti iz po 8 plavih čvorova od svojih N, S, E, i W susjeda (uz svoje "stranice"), kao i
- vrijednost iz po jednog plavog čvora od svojih dijagonalnih NW, SW, SE i NE susjeda (uz svoje "vrhove").

Analogno, za nove vrijednosti u (crvenim) čvorovima sljedećeg problema $P^{(4)}$, on treba

- vrijednosti iz po 4 crvena čvora od svojih N, S, E, i W susjeda, i
- vrijednost iz po jednog crvenog čvora od svojih dijagonalnih NW, SW, SE i NE susjeda.

Isti princip komunikacije vrijedi sve dok svaki procesor ima bar po jedan čvor odgovarajuće mreže, tj. do problema $P^{(k)}$.

Nakon toga, u još grubljim problemima, samo neki procesori učestvuju u komunikaciji (i računanju). Svaki od njih treba po jednu vrijednost od nekih, u općem slučaju, 8 procesora (ne nužno susjednih). Naravno, uz rubove, broj potrebnih vrijednosti pada.

Ovo nam je sasvim dovoljno da napravimo asimptotsku ("veliko $\mathcal{O}$") analizu cijene komunikacije i računanja.

Pogledajmo prvo V-ciklus koji počinje na nivou m. U nivoima od m do k, svaki procesor ima barem po jednu točku mreže. Nakon toga, na nivoima $k-1$ do 1, manje od cjelokupnog broja procesora ima aktivnu točku odgovarajuće mreže, pa neki procesori ne rade ništa.

Kao i ranije, neka je f vrijeme po flopu, α vrijeme za početak komunikacije i β vrijeme slanja po riječi (broju) u poruci.

Kad smo na nivou i u gornjem dijelu V-ciklusa, za $m \geq i \geq k$, potrebno vrijeme za taj nivo Time(i) je:

$$\text{Time}(i) = O(4^{i-k}) * f + O(1) * \alpha + O(2^{i-k}) * \beta.$$

Prvi član dolazi od broja operacija po procesoru, koji je proporcionalan broju čvorova po procesoru, drugi član dolazi od konstantnog broja komunikacija sa susjedima, a treći član je broj poslanih riječi.

Zbrajanjem svih članova za $i = k, \ldots, m$ dobivamo

$$\begin{aligned}\text{Time}(k\_m) &= O(4^{m-k}) * f + O(m-k) * \alpha + O(2^{m-k}) * \beta \\ &= O(n^2/p) * f + O(\log(n/p)) * \alpha + O(n/\sqrt{(}p)) * \beta.\end{aligned}$$

Uočimo da ovdje imamo efekt "površina prema volumenu", čim je $i \gg k$, kada svaki procesor komunicira tako da šalje samo svoje rubne podatke, a računa s mnogo više točaka no što ih posjeduje. Taj fenomen vidjeli smo kod simulacije skupa čestica kada je vrijeme računanja dominantno nad vremenom komunikacije.

Za nivoe $k \geq i \geq 1$, taj efekt nestaje, jer je broj transfera riječi za svaki procesor približno jednak broju računskih operacija:

$$\text{Time}(i) = O(1) * f + O(1) * \alpha + O(1) * \beta.$$

Zbrajanjem svih članova dobivamo:

$$\begin{aligned}\text{Time}(1\_(k-1)) &= O(k-1) * f + O(k-1) * \alpha + O(k-1) * \beta \\ &= O(\log(p)) * f + O(\log(p)) * \alpha + O(\log(p)) * \beta.\end{aligned}$$

Ukupno vrijeme za V-ciklus koji počinje na najfinijem nivou je:

$$\text{Time} = O(n^2/p + \log(p)) * f + O(\log(n)) * \alpha + O(n/\sqrt{(}p) + \log(p)) * \beta.$$

Slično dobivamo i ukupno vrijeme za V-ciklus koji počinje na nivou j, za $k \leq j \leq m$:

$$\text{Time}(j) = O(4^j/p + \log(p)) * f + O(j) * \alpha + O(2^j/\sqrt{(}p) + \log(p)) * \beta.$$

Ukupno vrijeme V-ciklusa koji počinje na nivou j, za $j < k$, je:

$$\text{Time}(j) = O(j) * f + O(j) * \alpha + O(j) * \beta.$$

Prema tome, ukupno vrijeme za potpuni multigrid (FMG), uz pretpostavku $n^2 \geq p$ (svaki procesor ima barem 1 čvor), je:

$$\begin{aligned}\text{Time} &= \sum_{j=1}^{m} \text{Time}(j) \\ &= O(n^2/p + \log(p) * \log(n)) * f + O((\log(n))^2) * \alpha \\ &\quad + O(n/\sqrt{(}p) + \log(p) * \log(n)) * \beta \\ &= O(N/p + \log(p) * \log(N)) * f + O((\log(N))^2) * \alpha \\ &\quad + O(\sqrt{(}N/p) + \log(p) * \log(N)) * \beta,\end{aligned}$$

gdje je $N = n^2$ broj nepoznanica. Ubrzanje za računske operacije obzirom na sekvencijalni algoritam koji zahtijeva $O(N)$ računskih operacija je $O(N/p + \log(p) *$

$\log(N)$), što je skoro savršeno kad je prvi član dominantan, tj. kad je $N \gg p$, ali se reducira na $\log(N)^2$ kad je $p = N$ (recimo u PRAM modelu).

U praksi, vrijeme provedeno u najgrubljoj mreži, kad svaki procesor ima ili 1 aktivan čvor ili ga uopće nema, je beznačajno za sekvencijalni procesor, ali je značajno za paralelno računalo, pa se zbog toga ozbiljno ruši efikasnost paralelnog algoritma. Na primjer, u članku "Analysis of the Multigrid FMV Cycle on large scale parallel machines", R. Tuminaro and D. Womble, SIAM J. Sci. Comp. v. 14, n. 5, 1993, autori su razmatrali razne varijante multigrida za nCUBE2 računalo s 1024 procesora. To računalo imalo je relativno nisku latenciju i širok pojas komunikacije obzirom na brzinu realnih računskih operacija. Za 64×64 nepoznanica, imamo samo 4 nepoznanice po procesoru u najfinijoj raspodjeli, (vrlo malo za tako "veliko" računalo), pa je efikasnost V-ciklusa bila samo 0.02, a FMG-a samo 0.008. U tom slučaju većina procesora je većinu vremena besposlena. Za mrežu 1024×1024 čvorova imamo 1024 nepoznanice po procesoru, pa je efikasnost V-cilusa 0.7, a FMG-a 0.42, što je vrlo razumno.

1.6. Usporedba paralelnih metoda za rješavanje diskretne Poissonove jednadžbe

Usporedimo

- SOR (Successive OverRelaxation s optimalnim parametrom w_{opt}),
- FFT (Fast Fourier Transform), i
- Multigrid

za rješavanje Poissonove jednadžbe na mreži $n \times n$ s $N = n^2$ nepoznanica. Ovdje nećemo biti pretjerano "sitničavi" u smislu da FFT-u odgovara da je $n = 2^m$, a multigridu da je $n = 2^m - 1$. To će se izgubiti u O notaciji. Pretpostavljamo da naše računalo ima p procesora, a svaka realna operacija traje f sekundi. Slanje svake poruke s k riječi traje $\alpha + k * \beta$ sekundi.

	#flops	#poruka	#poslanih riječi
SOR	$N^{3/2}/p$	$N^{1/2}$	N/p
FFT	$N * \log(N)/p$	$p^{1/2}$	N/p
$Multigrid$	$N/p + \log(p) * \log(N)$	$(\log(N))^2$	$(N/p)^{1/2} + \log(p) * \log(N)$

Dakle, SOR je sporiji nego druge dvije metode po svim parametrima. S druge strane, teško je reći ima li manje računskih operacija u FFT-u ili multigridu, jer broj iteracija u FMG-u ovisi o tome kolika se točnost želi, dok FFT daje uvijek punu točnost. Nadalje, multigrid očito šalje manje riječi nego FFT ili SOR.

S druge strane, nije jasno šalje li FFT ili multigrid manje poruka. Broj poruka je sporije rastuća funkcija od N no što je to broj operacija ili broj poslanih riječi i za dovoljno veliki N bit će manjeg reda veličine no druga dva člana. Ali, ako je cijena po poruci α velika, a p malen, nije jasno što je brže — FFT ili Multigrid, jer u tom slučaju O analiza nije dovoljna da bi to razlikovala (sve ovisi o implementaciji).