imnla dn

Tadej Petrič

May 15, 2023

Naloge sem reševal samostojno. Reševal sem v C++ z Eigen knjižnico za linearno algebro, 3D ploti za poročilo prve naloge pa so v pythonu (zato ker uporabljam linux in knjižnice za izrisovanje pogosto ne delujejo na Windowsu brez veliko dela, tako da bi vi imeli težave pri poganjanju kode), ampak je samo klic surface_plot() matrike, ki sem jo v C++ shranil v datoteko. Podobno sem python uporabil za pretvorbo .mat datoteke za 4. nalogo v .mtx format, ki ga lahko bere Eigen - že pretvorjene matrike prilagam. Slike so malo grdo razporejene po poročilu, ker imajo veliko belega prostora - če piše "Figure n" spadajo k prvi nalogi.

Datoteke se nahajajo v `https://drive.google.com/file/d/1Ol8bcOQcKSPhhljzpvcTBuyy1r5-WYz1/view?usp=share_link` Rešitve se nahajajo v `https://drive.google.com/file/d/10EhI3-JQRvuCHleHhmxTJcJDZ70lG9xX/view?usp=share_link` ali na github repozitoriju `https://github.com/tadejpetric/nla`

Vsi programi delajo na mojem računalniku (razen kjer rečeno eksplicitno). Da koda deluje je potrebno le nastaviti "include_directories" v CMakeLists na lokacijo Eigna ter pognati cmake (lahko se tudi odstrani -fopenmp v flagsih, kar izklopi multithreading ampak verjetno bolje dela na Windows). Če imate datoteke shranjene v drugačni strukturi map je potrebno spreminjati tudi location (npr pri nalogi 3). Pri nalogi 4 pa deluje, če poganjate iz iste datoteke (klic `nal4.exe` po compilu) ali pa iz druge datoteke (klic `nal4.exe /home/user/programming/nla/nal4/` oziroma pot do tam kjer je shranjena train.mtx in test.mtx - v retrospektivi bi moral tako naresti tudi tretjo nalogo ampak navodila za namestitev pišem prepozno).

1 Naloga 1

Naj bo $U(x,y) = U_{x,y}$ matrika

$$\begin{aligned}
\Delta u(x,y) &= U_{xx} + U_{yy} \\
h \cdot U_x &= U(x) - U(x+1) \\
h^2 \cdot U_{xx} &= U(x-1,y) - 2U(x,y) + U(x+1,y) \\
h^2 \cdot U_{yy} &= U(x,y-1) - 2U(x,y) + U(x,y+1)
\end{aligned}$$

Če to damo skupaj, dobimo

$$\Delta U(x,y) + k(x,y)U(x,y) = 1$$

$$h^2 = U(x-1,y) + U(x+1,y) + U(x,y-1) + U(x,y+1) + (h^2 \cdot k(x,y) - 4)U(x,y)$$

Vemo, da je $U(i,j) = 0$ na $\partial[-1,1]^2$ oziroma $\{0,n\} \times [0,n]$ in obratno. V matriki za relacije računamo ostale vrednosti.

Problem preuredimo v enačbo $Ax = b$ kjer je x $n \cdot n$ vektor, ki predstavlja rešitev $u(i,j) = x_{n \cdot i+j}$. Relacije od zgoraj sestavimo v matriko; stran b je kar h^2 v vseh komponentah. Diagonalni elementi so ponavadi 4, razen ko upoštevamo kaznovalni parameter k.

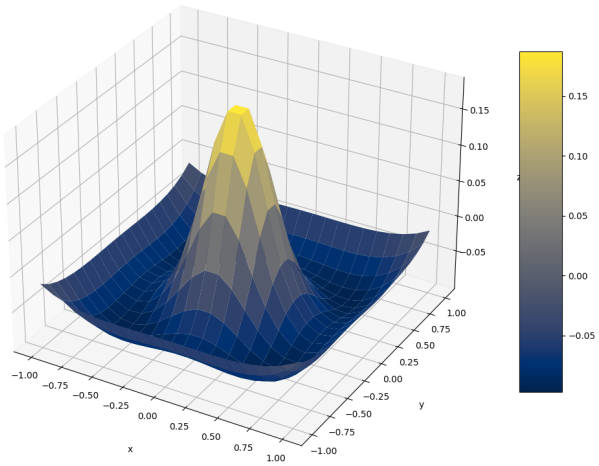


Figure 1: n 20, k 50

Višji kot je n, boljšo rešitev dobimo, saj vzorčimo bolj drobne delce (in teoretična izpeljava deluje le, ko gre $n \to \infty$). Do numeričnih težav ne pride za smiselne vrednosti n (na primer pod nekaj tisoč), če bi pa pretiravali bi pa lahko prišlo do težav.

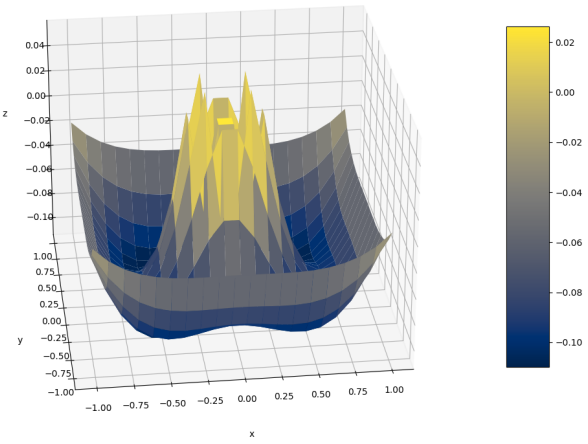


Figure 2: n 20, k 500

Kaznovalni parameter poskrbi, da so vrednosti na krogu približno enake 0, saj dobimo enačbo oblike $k \cdot u = 1 - \Delta$, kar je približno $h^2 ku = O(u)$. Torej, da poskrbimo, da je u majhen, more biti $h^2 k >> 0$. To lahko naredimo tako, da (neoptimalno) uporabimo $k \approx n^3$ ali pa $k \approx C \cdot n^2$.

Višji kot je k boljši približek dobimo (lahko uporabimo tudi $k = \infty$, vendar je teoretična analiza potem nerodna). Če je ta dovolj velik (kot v prejšnjih ocenah) je matrika šibko diagonalno dominantna (in nerazcepna) in lahko uporabimo iterativno metodo. Za dan problem dobro deluje Jacobijeva metoda, ki konvergira.

2 Naloga 2

Nedokončana. V datoteki spišem D-Lanczos algoritem (ki deluje) z LU razcepom in ga uporabim, da popravim na QR razcep.

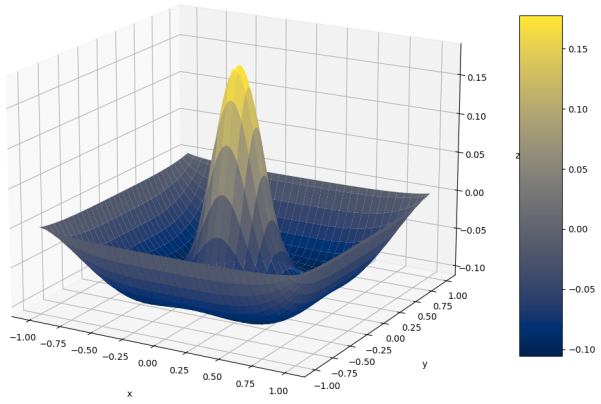


Figure 3: n 101, k 100

Če pišemo podobno izpeljavo kot za LU D-Lanczos, vidimo

$$\begin{aligned} x_k &= x_0 + V_k T_k^{-1} \|r_0\| e_1 \\ x_0 + V_k R_k^{-1} Q_k^\top \|r_0\| e_1 & \\ P_k R_k &= V_k \\ z_k &= Q_k^\top \|r_0\| e_1 \end{aligned}$$

Če napišemo Q_k kot $G_1^\top \cdots G_k^\top$ (kjer so G_i Givensove rotacije) dobimo zadnjo vrstico

$$Q_k^\top = G_k \cdots G_1$$

Torej je $\zeta_k = s \cdot \zeta'_{k-1}$. Vendar pa se v zaradi Givensove rotacije spremeni tudi $\zeta_{k-1} = c \cdot \zeta'_{k-1}$.

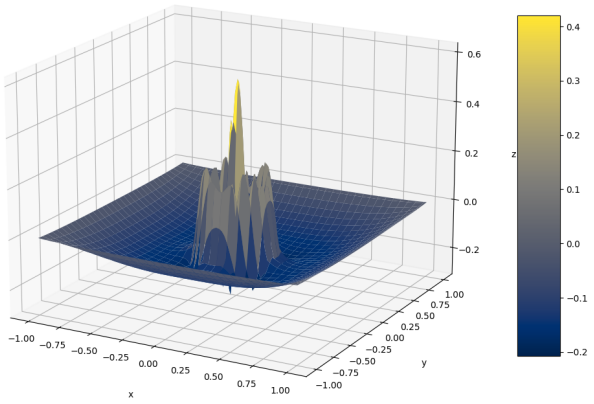


Figure 4: n 101, k 1000

Podobno analizo delamo pri R_k. Vemo, da je pri QR razcepu od tridiagonalne matrike R zgornja trikotna matrika z le tremi neničelnimi naddiagonalami, torej lahko element p_k napišemo kot $p_k = (v_k - \lambda_0 p_{k-2} - \lambda_1 p_{k-1})/\lambda_2$, kjer je λ_2 diagonalni element matrike v zadnjem stolpcu, λ_1 naddiagonalni in λ_2 nad naddiagonalnim. Tukaj je spet potrebno popravljati koeficient še v naslednji iteraciji, saj ga naslednja givensova rotacija spremeni, vendar lahko to poskrbimo brez da hranimo dodatni vektor (hranimo le koeficiente in givensove rotacije, kar je poceni). Potrebno je torej hraniti p_k, p_{k-1}, p_{k-2}, v_this, v_last, A, x, b in konstantno mnogo številskih parametrov.

Moja implementirana metoda ne deluje, ker sem se jo lotil pisati nespametno ter potem raje reševal druge naloge, osnutek je pa končan v kodi (do permutacije vrstic natančno, saj bi bilo treba pravilno upoštevati rotacije).

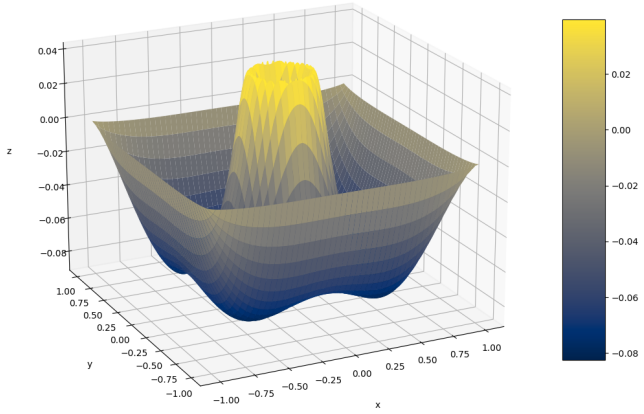


Figure 5: n 200, k 300

3 Naloga 3

Vse metode uporabljamo z toleranco 1e-6 in maksimalnim številom korakov 500. Izhod programa je priložen v datoteki `output.txt` (saj se sicer izvaja precej dolgo časa). Namesto nekaterih (neimplementiranih v knjižnici) Matlabovih metod sem uporabil ostale Eigen iterativne metode (idrs, idrstabl, idrstab, bicgstabl).

Najlepša matrika je gridgena. Na njej delujejo vse metode, saj je simetrična in SPD. Uporabimo lahko tudi dobro predpogovjevanje, saj je cholesky razgradljiva. Še posebej uporabne so metode idrstabl, bicgstabl in gmres s predpogojevanjem choleskega. Zaradi izjemno hitre konvergence (v le 15 iteracijah) parameter ponovnega zagona nima velikega vpliva. Predpogojevanje pomaga mnogim metodam: metodi minres, idrs, idrstabl in bistab delajo najboljebolje z diagonalnim predpogojevanjem; gmres, cg pa s predpogojevanjem choleskega.

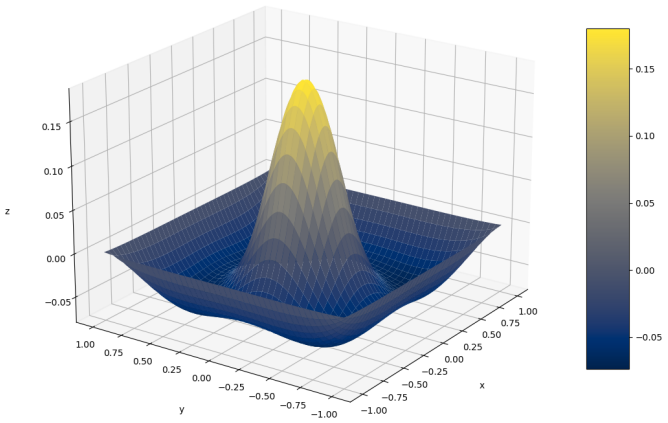


Figure 6: n 301, k 50

Idrstabl z diagonalnim predpogojevanjem deluje hitreje kot metoda SparseLU.

Slabše se obnaša c-63. Je sicer simetrična, ni pa pozitivno definitna ali Cholesky razgradljiva. Metode s privzetimi nastavitvami so počasne. Najbolje delata idrstabl in bicgstabl, sprejemljivo pa kovnergira tudi gmres brez ponovnega zagona, kjer potrebuje 254 iteracij z 64 restart in 103 iteracij brez ponovnega zagona. Za manjše vrednosti restarta ne konvergira znotraj 500 iteracij. Če dodamo diagonalno predpogojevanje se konvergenca veliko pospeši. Zelo dobro delajo vse metode (razen minres zaradi tehničnih razlogov). Če uporabimo predpogojevanja lahko tudi rešimo hitreje kot SparseLU z vsemi metodami.

RFDevice ima za b matriko z devetimi stolpci. Stolpca 0 in 4 sta slabša od ostalih, za njiju ne deluje dobro nobena iterativna metoda. Pri ostalih stolpcih gmres prez predpogojevanja pride do rešitve v petih korakih. Sprejemljive metode so tudi bicgstab in idrs in idrstabl. Predpogojevanja le poslabšajo rezul-

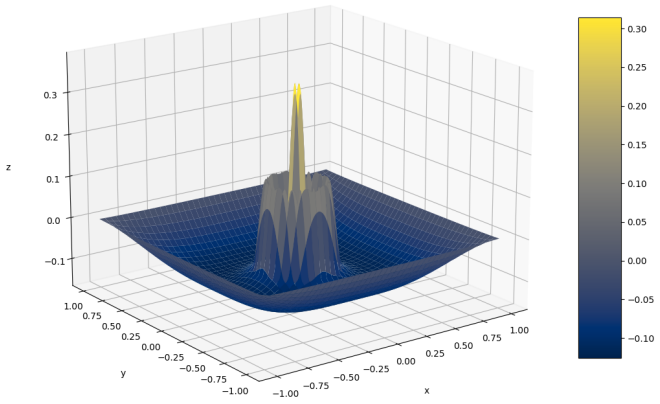


Figure 7: n 301, k 1000

tate.

4 Naloga 4

Da določimo optimalen α poženemo algoritem za različne α s privzeto toleranco (okoli 1e-15). Spremljamo, koliko korakov je bilo potrebnih za konvergenco. Nato preverimo odstopanje rešitve od neregularizirane rešitve.

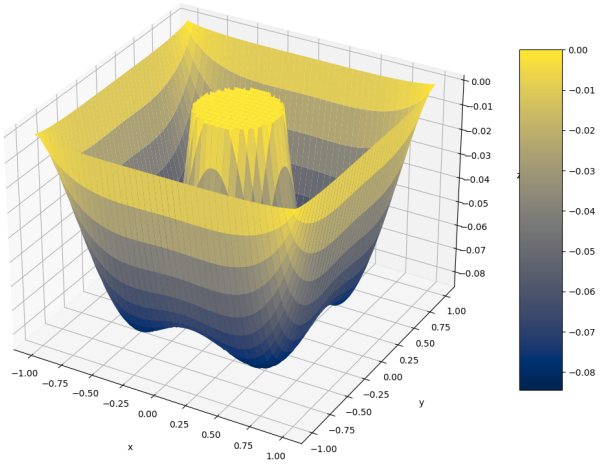
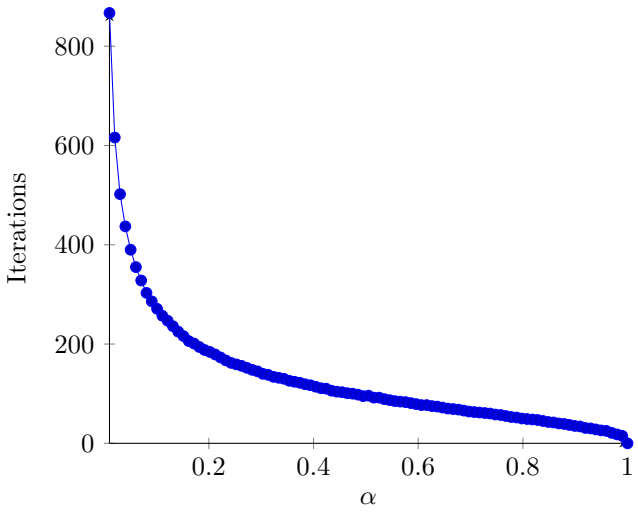


Figure 8: n 301, k 1 000 000



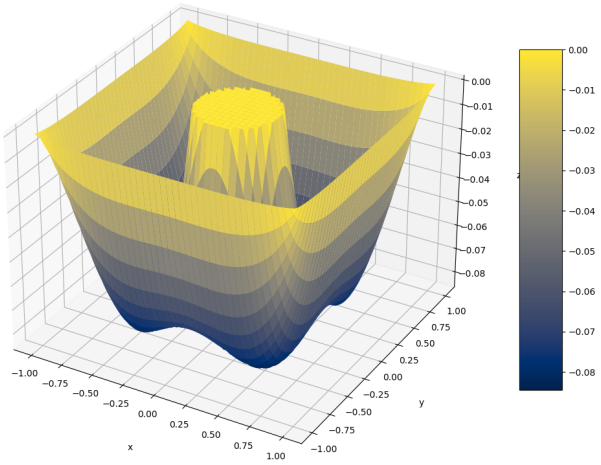
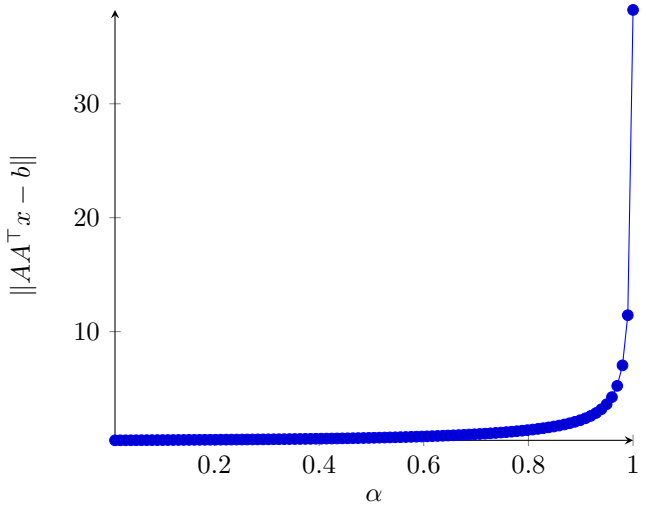


Figure 9: n 301, k 1 000 000 000



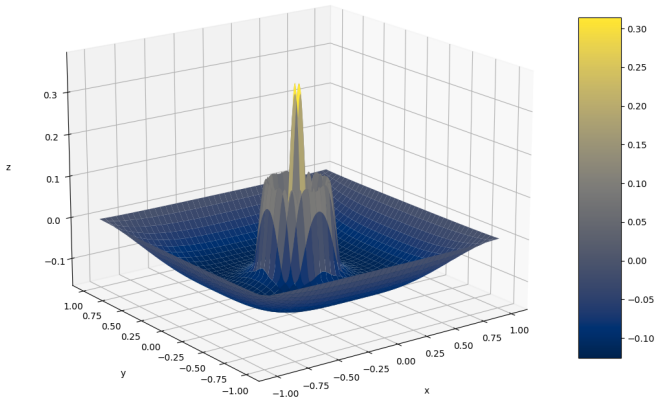


Figure 10: n 301, k 1000

Iz grafa razberemo, da je za vrednost $\alpha = 0.6$ rešitev zelo blizu dejanske rešitve, število iteracij pa zadovoljivo nizko. Računamo matriko $X^\top A^{-1} X$. Zaradi učinkovitosti v spomin shranimo le en stolpec končnega rezultata na enkrat in ga shranimo v datoteko (column major format, tako da izgleda, kot da shranimo transponiranko).

Računanja stolpca i se lotimo tako, da preslikamo e_i z X, kar nam da x_i. Nato rešimo sistem $Ay = x_i$ z metodo konjugiranih gradientov. Rešitev y preslikamo z X, da dobimo i-ti stolpec končne matrike.

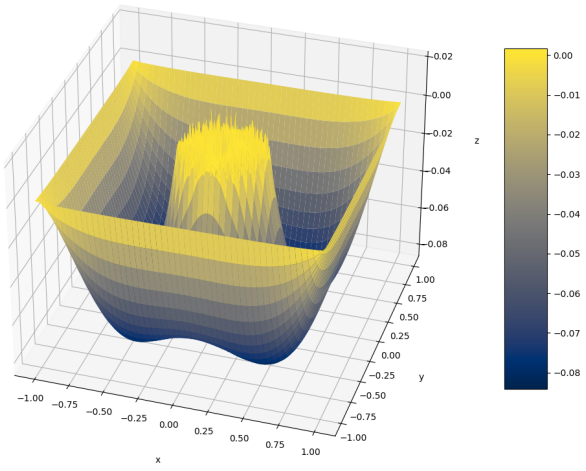


Figure 11: n 600, k 100000

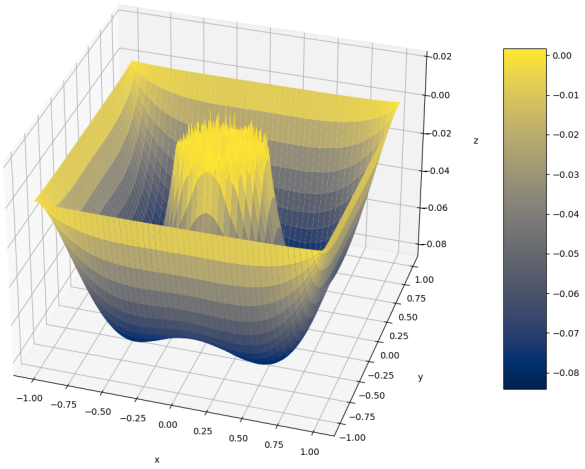


Figure 12: n 301, k 1e6, Jacobi s 3000 iteracij