SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2032

ANALIZA I MODELIRANJE OBJEKATA U SLICI 2D GAUSSOVIM PROFILOM

Matija Jurišić

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2032

ANALIZA I MODELIRANJE OBJEKATA U SLICI 2D GAUSSOVIM PROFILOM

Matija Jurišić

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

DIPLOMSKI ZADATAK br. 2032

Pristupnik:	**Matija Jurišić (0036494683)**
Studij:	Elektrotehnika i informacijska tehnologija
Profil:	Elektroničko i računalno inženjerstvo
Mentor:	prof. dr. sc. Davor Petrinović

Zadatak: **Analiza i modeliranje objekata u slici 2D Gaussovim profilom**

Opis zadatka:

U okviru diplomskog rada potrebno je napraviti obradu dvodimenzionalne slike u kojoj je potrebno identificirati višestruke objekte njihovim modeliranjem pomoću multivarijantne normalne (Gaussove) razdiobe. Parametri modela koje je potrebno estimirati su vektor očekivanja, tj. očekivana pozicija objekta na slici, kovarijancijska matrica koja opisuje oblik objekta, i njegov intenzitet, tj. amplitudu. Potrebna obrada slike već je realizirana korištenjem MATLAB programskog okruženja, a zadatak ju je realizirati u jeziku C/C++. U realizaciji je preporučeno korištenje OpenCV biblioteke otvorenog koda s primjenom u području računalnog vida. Istražiti i mogućnost implementacije u okruženju za paralelno programiranje CUDA, čime se omogućuje učinkovitije rješavanje složenijih računalnih problema na grafičkim procesorima. Točnost i brzinu rada sustava potrebno je usporediti s referentnom implementacijom u Matlabu na sintetičkim i stvarnim slikama.

Rok za predaju rada: 30. lipnja 2020.

ZAHVALA

Zahvaljujem se mentoru prof. dr. sc. Davoru Petrinoviću na razumijevanju, trudu, pomoći i vodstvu prilikom izrade ovog diplomskog rada. Također zahvaljujem svojim roditeljima, bratu, te svima iz obitelji i prijateljima na pruženoj podršci i strpljenju tijekom cijeloga studija.

SADRŽAJ

1. Uvod

U ovom diplomskom radu napravljena je obrada dvodimenzionalne slike u kojoj se identificiraju višestruki objekti njihovim modeliranjem pomoću multivarijantne normalne (Gaussove) razdiobe. Estimirani su parametri modela koji sadrže vektor očekivanja, tj. očekivanu poziciju objekta na slici, kovarijancijsku matricu koja opisuje oblik objekta, i njegov intenzitet, tj. amplitudu. Potrebna obrada slike već je bila realizirana korištenjem MATLAB programskog okruženja, a povodom zadatka ovog rada realizirana je u jeziku C. Rad je strukturiran tako da je prvo opisana glavna funkcija, a zatim sve pomoćne korištene funkcije. Za integriranje C koda u MATLAB, kod je bilo potrebno konfigurirati u MEX datoteku. Na kraju je uspoređena točnost i brzina rada sustava s referentnom implementacijom u MATLABU.

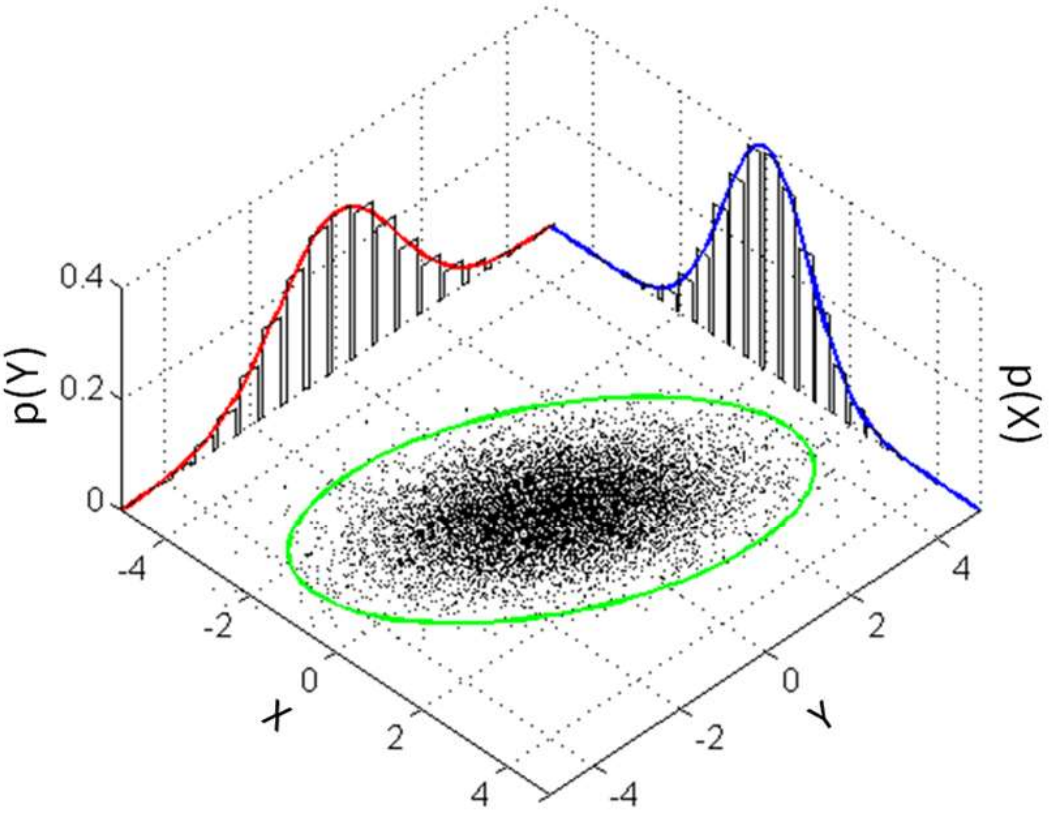
2. Multivarijantna normalna (Gaussova) razdioba

U teoriji vjerojatnosti i statistike, multivarijantna normalna ili Gaussova razdioba je poopćenje univarijantne normalne razdiobe za više dimenzija. Neki vektor ima multivarijantnu normalnu razdiobu ako svaka linearna kombinacija od njegovih komponenata ima jednodimenzijsku ili univarijantnu normalnu razdiobu. Takva vrsta razdiobe često se koristi za opisivanje i aproksimaciju bilo kojeg skupa koreliranih realnih vrijednosti slučajnih varijabli, od kojih se svaka grupira oko srednje vrijednosti.

Multivarijantna Gaussova razdioba se računa kada je simetrična kovarijantna matrica pozitivna. U tom slučaju razdioba ima gustoću:

$$f_x(x_1, ..., x_k) = \frac{exp(-\frac{1}{2}(x-\mu)^T \sum^{-1}(x-\mu))}{\sqrt{(2\pi)^k |\sum|}}$$

gdje je x realni vektor sa k stupaca, μ srednja vrijednost, a $|\sum|$ determinanta od $\sum$. Dio jednadžbe $\sqrt{(x-\mu)^T \sum^{-1}(x-\mu)}$ se naziva Mahalanobisova udaljenost i predstavlja udaljenost između x i μ. Na slici 2.1 je prikazan primjer funkcije vjerojatnosti za multivarijantnu normalnu distribuciju.



Slika 2.1: Primjer funkcije vjerojatnosti sa 3σ elipsom, dvije granične razdiobe i dva histograma

3. Programsko rješenje funkcije za estimaciju inicijalnih parametara 2D Gauss modela

3.1. Općenito o funkciji

Ova funkcija služi za dobivanje rezultata pojedinačnih estimiranih modela ovisno o zadanim ulaznim empirijskim podacima (x_i, y_i, z_i). Funkcija kao ulazne parametre prima stupac z, dvostupčanu matricu v, binarnu matricu rub, z_length koja označava duljinu stupaca podataka, te dxy, $cent\_iter$ i $debu$, čija će svrha naknadno biti objašnjena. Prvi ulazni stupac z sadrži podatke na zadanoj rijetkoj i raštrkanoj mreži $z_i = f(x_i, y_i)$, tj. sadrži dvodimenzionalnu funkciju gustoće vjerojatnosti pomnoženu s nepoznatom skalom. Dvostupčana matrica v sadrži parove vrijednosti domene (x_i, y_i), i time definira domenu funkcije gustoće vjerojatnosti. Nadalje, rub je binarna matrica rubnih uzoraka izdvojene regije koja označava parove (x_i, y_i) koji se nalaze na rubu izdvojene regije i time definiraju obuhvat empirijskog modela. Zadnje spomenuta tri parametra sadrže svaki po jedan podatak. dxy označava mjeru integracije radi pretvore u histogram, a $cent\_iter$ je broj iteracija vanjske petlje za iterativno poboljšanje modela, odnosno matrice kovarijance C i centroida. $debu$ se u MATLAB verziji funkcije koristio za prikaz slika ako mu je vrijednost bila jednaka jedan, a u C implementaciji funkcije se ne koristi te mu je vrijednost postavljena u nulu.

Uz navedene parametre postoji i zadnji ret_array, koji se pomoću pokazivača koristi kao izlaz funkcije. On je predstavljen kao vektor podataka u koji se onda spremaju razni izračunati rezultati iz funkcije. Na početku vektora nalazi se matrica $mode$ koja je spremljena red po red i koja u svakom retku sadrži parametre koreliranog 2D Gauss modela [*s1 s2 ro x0 y0*]. Ukupno ima $cent\_iter + 2$ retka koji redom opisuju:

redak 1: inicijalni model temeljen na prvom i drugom momentu,

redak 2: skalirani inicijalni model s faktorom zahvata k, jednakim kao za ulazni stupac z,

redak 3: korigirani model s optimalnom matricom kovarijance za isti centroid korišten u prva dva modela, minimizira pogrešku za fiksiranu matricu kovarijance,

redak 4 ... redak $cent\_iter + 2$**:** rekurzivno optimirani modeli, u svakoj iteraciji ponovno se računa optimalna matrica kovarijance za novi centroid koji je dobiven u prošloj iteraciji.

Retci od trećeg nadalje računaju se samo ako je $cent\_iter \geqslant 1$. Nakon matrice, u izlazni vektor se sprema $skala$, tj. faktor optimalne skale koji množi Gaussovu funkciju i predstavlja njezin integral. Iza faktora se nalazi ukupna pogreška u odnosu na empirijske podatke u dB. Ta pogreška u kodu je nazvana sa err, i računa se za svaku od $cent\_iter+2$ iteracija kao i $skala$. Također, izlaz sadrži ukupnu težinsku kvadratnu pogrešku argumenta eksponencijalne funkcije ili $wearg$. On aproksimira ukupnu kvadratnu pogrešku modela u odnosu na z za svaku iteraciju. Kao sljedeća vrijednost na izlazu se nalazi $iter\_detC$, tj. maksimalni broj iteracija za konvergenciju determinante C za iterativni postupak baziran na tri jednadžbe. Na kraju ret_array vektora je smješten broj iteracija petlje za konvergenciju centroida za trenutačnu optimalnu matricu kovarijance, $iter\_cent$. Zadnje dvije varijable se također računaju za svaku iteraciju i sadrže $cent\_iter + 2$ podataka na izlazu.

Za dobivanje korigiranog modela, tj. računanje trećeg retka na dalje u izlaznim podacima, z_length ulazni parametar mora imati vrijednost sedam ili više. Traženi model ima šest slobodnih parametara pa je u tom slučaju zadovoljen minimalni broj uzoraka Gaussove funkcije (x_i, y_i, z_i). Tada je moguće naći model koji minimizira minimalno kvadratno odstupanje od zadanih uzoraka z_i u domeni vrijednosti eksponencijalne funkcije. Sama optimizacija se provodi u domeni argumenta eksponencijalne funkcije uz korištenje pogodno odabrane težinske funkcije.

3.2. Implementacija glavne funkcije

Na početku je potrebno sve vrijednosti sa ulaza z koje su negativne ili jednake nuli pretvoriti u minimalnu pozitivnu vrijednost, pod pretpostavkom da se radi o Gaussovoj funkciji bez pomaka. Kako bi se izdvojio dio ulaznih podataka, radi se procjena faktora k pomoću kojeg se onda zahvaća $+/-k\sigma$ podataka. Faktor se računa na osnovu kvocijenta najveće i rubne vrijednosti ulazne empirijske funkcije vjerojatnosti.

Sljedeće je potrebno odrediti inicijalnu skalu ulaznih podataka, tako da je nakon denormalizacije integral empirijske funkcije preko izdvojene domene jednak očekivanoj vrijednosti pro. Prije toga je potrebno izračunati pro koji je jednak vjerojatnosti da se slučajna varijabla nalazi unutar $+/-k\sigma$. Za dobivanje inicijalne skale, u računu se koristi i ulazni parametar mjere integracije dxy kako bi se od gustoće dobila vjerojatnost. Kada je poznata skala, provodi se denormalizacija ulaznih podataka i podaci se spremaju u zn.

Sa dosad dobivenim podacima moguće je provesti inicijalizaciju modela i izračun potrebnih momenata. Nalaze se prvi poopćeni momenti $x0$ i $y0$ gdje se za težinu koriste empirijski podaci uz potenciju pot koja je u ovoj implementaciji jednaka jedan. Nakon toga se nad momentima provodi normalizacija.

```
double *x0_data, *y0_data;
x0_data = mxCalloc(z_length, sizeof(double));
y0_data = mxCalloc(z_length, sizeof(double));
double *x0;
double *y0;
x0 = mxCalloc(1, sizeof(double));
y0 = mxCalloc(1, sizeof(double));
double zn_pot_sum = 0;
for (i = 0; i < z_length; i++) {
        x0_data[i] = v1[i] * pow(zn[i], pot);
        y0_data[i] = v2[i] * pow(zn[i], pot);
        x0[0] += x0_data[i];
        y0[0] += y0_data[i];
        zn_pot_sum += pow(zn[i], pot);
}

mxFree(x0_data);
mxFree(y0_data);

x0[0] = x0[0] / zn_pot_sum;
y0[0] = y0[0] / zn_pot_sum;
```

Izračunati prvi momenti se odma koriste za dobivanje $v0$ ili odstupanja točaka od očekivane vrijednosti. Odstupanje se određuje tako da se od parova vrijednosti domene (x_i, y_i) sadržanih u ulaznom parametru v oduzme $x0$, odnosno $y0$. Drugi momenti se računaju obzirom na parove (x_i, y_i) kojima se uklanja prvi moment i time dobiva estimacija varijanci. Nakon toga je potrebno naći očekivanje i iz njega i drugih momenata odrediti parametar ro. Nad drugim momentima $s1$ i $s2$ se tada može primijeniti teorij-

ski korekcijski faktor skaliranja.

3.3. Inicijalni model

Uz sve parametre poznate, moguće je formirati vektor s parametrima inicijalnog modela $mode\_init$. Koristeći inicijalni model računa se inicijalni interpolirani normalizirani Gaussov model s teorijski skaliranim poluosima za zadanu regiju. Za to se koristi funkcija *gauss2D_fun_sparse*, koja je detaljnije opisana u 4. poglavlju.

```
double *mode_init;
mode_init = mxCalloc(5, sizeof(double));
mode_init[0] = s1;
mode_init[1] = s2;
mode_init[2] = ro;
mode_init[3] = x0[0];
mode_init[4] = y0[0];

double *z1ok;
z1ok = mxCalloc(z_length, sizeof(double));
gauss2D_fun_sparse(mode_init,v1,v2,0,z_length,z1ok);
```

Nakon povratka iz pomoćne funkcije sada je moguće izračunati prvi podatak optimalne amplitudne skale pomoću vrijednosti dobivene iz funkcije. $skala[0]$ se dobiva amplitudnim skaliranjem s optimalnim iznosom amplitudne skale koji osigurava najmanju kvadratnu razliku između ulaznih podataka z i njihove predikcije.

```
double z1ok_mul = 0;
double z1okz_mul = 0;
for (i = 0; i < z_length; i++) {
        z1ok_mul += z1ok[i] * z1ok[i];
        z1okz_mul += z1ok[i] * z2[i];
}
skala[0] = z1okz_mul / z1ok_mul;
```

Izračunati parametri inicijalnog modela $mode\_init$ iskorišteni su za popunjavanje prvog reda matrice $mode$. Kako bi se dobio prvi podatak pogreške inicijalnog modela err koristi se razlika između empirijskih podataka i inicijalnog modela.

```
double *raz;
raz = mxCalloc(z_length, sizeof(double));
//nastavak na sljedecoj stranici
```

```
for (i = 0; i < z_length; i++)
        raz[i] = z2[i] - z1ok[i] * skala[0];

double raz2_sum = 0;
double z2_sum = 0;
for (i = 0; i < z_length; i++) {
        raz2_sum += pow(raz[i], 2);
        z2_sum += pow(z2[i], 2);
}
err[0] = 10 * log10(raz2_sum / z2_sum);
```

3.4. Skalirani inicijalni model

Problem prvog modela je što nema isti zahvat k kao što imaju empirijski podaci. Zbog toga inicijano estimirani model nema pravu skalu sigmi, te ga je potrebno dodatno korigirati drugim modelom. U izračunu drugog modela iterativno se skaliraju estimirani parametri koreliranog modela tako da se nađe skala kod koje je $k\sigma$ faktor prostornog zahvata modela jednak empirijskom $k\sigma$ faktoru zahvata određenom iz ulaznih podataka. Inicijalna pretpostavljena vrijednost skala sigmi $corr\_term$ iznosi jedan. U Nct_iter iteracija, čiji je broj u ovoj implementaciji postavljen na sedam, radi se korekcija skale $s1$ i $s2$. Za to je potrebno naći maksimalnu vrijednost normaliziranog skaliranog modela pozivanjem *gauss2D_fun_sparse* funkcije sa prvim poopćenim momentima $x0$ i $y0$ kao ulaznim argumentima. Sličnim postupkom se nalazi i rubna vrijednost normaliziranog skaliranog modela $z1\_rub$, tako da se pomoćna funkcija poziva uz ulazne vrijednosti (x_i, y_i) parova koji se nalaze na rubu izdvojene regije.

```
m = 0;
double *v_rub_1;
double *v_rub_2;
v_rub_1 = mxCalloc(v_rub_length, sizeof(double));
v_rub_2 = mxCalloc(v_rub_length, sizeof(double));
for (j = 0; j < z_length; j++) {
        if (rub[j] != 0) {
                v_rub_1[m] = v1[j];
                v_rub_2[m] = v2[j];
                m++;
        }
}       //nastavak na sljedecoj stranici
```

```
double *z1_rub_data;
z1_rub_data = mxCalloc(z_length, sizeof(double));
gauss2D_fun_sparse(mode_corr,v_rub_1,v_rub_2,0,v_rub_length,
                   z1_rub_data);
```

Uz to je potrebno izračunati srednju vrijednost $z1\_rub\_data$ i time je dobiven $z1\_rub$. Sljedeće se procjenjuje faktor $kmod$ koji govori da je $+/-k\sigma$ modela zahvaćeno unutar korisnog izdvojenog dijela domene.

```
double kmod;
kmod = sqrt(-2 * log(z1_rub / z1_max[0]));
```

Koristeći izračunate vrijednosti moguće je korigirati multiplikativni faktor $corr\_term$ za skaliranje $s1$ i $s2$, kako bi novi $k\sigma$ faktor modela bio još bliži empirijskom faktoru zahvata ulaznih podataka k.

```
corr_term = (kmod / k) * corr_term;
```

Time je završen iterativni postupak i moguće je formirati konačni vektor parametara skaliranog koreliranog modela koji popunjava drugi red matrice $mode$. Izračunati vektor parametara koreliranog modela se koristi za dobivanje normaliziranog Gaussovog modela s optimalno skaliranim poluosima pozivanjem funkcije *gauss2D_fun_sparse* za domenu definiranu ulazom v. Ponovni izračun optimalne amplitudne skale, tj. $skala[1]$ se dobiva na isti način kao i $skala[0]$, samo što se sada koriste novoizračunati podaci $z1ok$ iz pomoćne funkcije. Isto tako se ponovno računa razlika empirijskih podataka i inicijalnog modela, te s dobiva nova pogreška korigiranog modela $err[1]$ koja je izražena u dB. Na kraju se sigme $s1$ i $s2$ ažuriraju sa korekcijskim članom skaliranja kako bi se mogle koristiti za inicijalizaciju preostalih modela.

3.5. Rekurzivno optimirani modeli

Treća faza estimacije se provodi ako se traže iterativno rafinirani modeli i ako je broj uzoraka domene dovoljan za nalaženje šest nepoznanica. U te nepoznanice spadaju rezidualna skala, centroid $(x0, y0)$ i tri člana matrice kovarijance. Za ispunjenje uvjeta treće faze duljina ulaznih stupaca z_length mora biti veća od šest i broj iteracija vanjske petlje $cent\_iter$ mora biti barem jedan. U trećoj fazi optimizacija se provodi u domeni argumenta eksponencijalne funkcije. Ulazni empirijski podaci se zbog toga moraju logaritmirati. Uz to, potrebno je uvesti i dodatnu težinsku funkciju zbog neliearnosti eksponencijalne funkcije. Optimizacija u domeni argumenta omogućuje da

se optimalna inverzna matrica kovarijance može naći kao analitičko rješenje linearnog problema u tri nepoznanice $Ci11$, $Ci12$ i $Ci22$ za zadani centroid $(x0, y0)$. Nakon nalaženja optimalne matrice C, radi se dodatna korekcija centroida kojom se za fiksiranu matricu kovarijance nalazi rješenje koje će dodatno smanjitit težinsku kvadratnu pogrešku argumenta. Za korekciju je potrebno pronaći analitičke gradijente težinske kvadratne pogreške eksponencijalnog argumenta po $x0$ i $y0$, te pripadajuću 2x2 Hessian matricu. Oni se onda koriste za pronalazak optimuma pomoću optimizacijskog postupka koji konvergira u određenom broju iteracija. Time se dobiva novi model koji je uvijek bolji od prethodnog prema korištenom kriteriju težinske kvadratne pogreške argumenta.

Treća faza započinje denormalizacijom empirijskih ulaznih podataka sa estimacijom skale drugog modela. Dodatnim množenjem sa ulaznim parametrom mjere integracije dxy se dobivaju diskretne vjerojatnosti svakog (x_i, y_i) para ulaznih vrijednosti.

```
for (i = 0; i < z_length; i++)
        zn[i] = (z2[i] / skala[1]) * dxy;
```

Sljedeće se formira težina za optimizaciju u domeni argumenta w. Ona je određena kvadratom linearne vrijednosti uzorka uz uključenu skalu i normalizacijski faktor Gaussa. Tako odabrana težina daje predikciju kvadratne pogreške modela u domeni vrijednosti eksponencijalne funkcije. Uz to je potrebno izračunati inicijalni argument eksponencijalne funkcije u logaritamskoj domeni, pri čemu se koristi logaritam gustoće koji se dobiva dijeljenjem histograma sa površinom ispod svakog stupca. Također se uklanja prvi dio nazivnika iz faktora normalizacije.

```
double *lz0;
lz0 = mxCalloc(z_length, sizeof(double));
for (i = 0; i < z_length; i++)
        lz0[i] = log(zn[i] / dxy + DBL_EPSILON) + log(2 * M_PI);
```

Ovisno u vrijednosti parametra broja iteracija $cent\_iter$ ulazi se u petlju u kojoj se iterativno traži korekcija matrice kovarijance i centroida. Ukoliko vrijedi da je $cent\_iter \geqslant 1$, u petlju se ulazi $cent\_iter$ puta. Petlja započinje sa izračunom drugog dijela faktora normalizacije u nazivniku koji je jednak drugom korijenu matrice kovarijance modela, pa se za njega koristi inicijalna estimacija skaliranog modela iz drugog retka izlaza.

```
double sdetC0;
sdetC0 = s1 * s2 * sqrt(1 - pow(ro,2));
```

Nakon toga je potrebno naći maksimum od vektora $lz0$ kako bi se kasnije s njime moglo osigurati da argument eksponencijalne funkcije uvijek bude negativan ili nula.

```
double *lz;
lz = mxCalloc(z_length, sizeof(double));
for (i = 0; i < z_length; i++) {
        if (-lz0_max < log(sdetC0))
                lz[i] = lz0[i] - lz0_max;
        else
                lz[i] = lz0[i] + log(sdetC0);
}
```

Inverzna matrica kovarijance Ci za inicijalni model računa se direktno na osnovu parametara polaznog koreliranog modela. Uz nju se u daljnjem računu koriste i relativne pozicije ulaznih uzoraka u odnosu na trenutačni centroid dx i dy, koje se dobivaju oduzimanjem $x0$, tj. $y0$ od ulaznih stupaca v. Koristeći dobivene parametre moguće je pronaći argument Gaussa za inicijalni model.

```
double *arg_init;
arg_init = mxCalloc(z_length, sizeof(double));
for (i = 0; i < z_length; i++)
        arg_init[i] = -((Ci11 * pow(dx[i],2)) / 2 + Ci12 * dx[i] *
                        dy[i] + (Ci22 * pow(dy[i],2)) / 2);
```

Razlika empirijskog argumenta i argumenta inicijalnog modela se sprema u vektor $como$ kao razlika između lz i arg_init. Prije iteriranja podataka u nove varijable se spremaju $sdetC0$ i sumarna kvadratna težinska pogreška modela u logaritamskoj domeni za inicijalni model. Također, u prvoj korekcijskoj iteraciji pohranjuje se težinska pogreška argumenta eksponencijalne funkcije.

```
sdetC[0] = sdetC0;
for (i = 0; i < z_length; i++)
        logerr[0] += w[i] * pow(como[i],2);

if (i_cent_iter == 2)
        wearg[1] = logerr[0];
```

Uz postojeće članove inverzne matrice kovarijance Ci, kao nova nepoznanica dodaje se nepoznati vetikalni pomak targeta i na taj način se pronalazi rješenje koje daje optimalni pomak targeta $z0$ za najmanju težinsku kvadratnu pogrešku argumenta. Time se formira novi sustava od četiri jednadžbe koji daje optimalno rješenje za Ci i $z0$. U

proširenoj matrici A, lijevi gornji dio jednak je polaznoj 3x3 matrici sustava jednadžbi za tri nepoznanice Ci, a zadnji novi stupac i redak su vezani uz varijablu $z0$.

$$A4 = \begin{bmatrix} dx^4w & 2dx^3dyw & dx^2dy^2w & -2dx^2w \\ 2dx^3dyw & 4dx^2dy^2w & 2dxdy^3w & -4dxdyw \\ dx^2dy^2w & 2dxdy^3w & dy^4w & -2dy^2w \\ -2dx^2w & -4dydyw & -2dy^2w & 4w \end{bmatrix}$$

Isto tako desni stupac linearnog sustava jednadžbi b je nadopunjen sa zadnjim retkom koji iznosi $(4*act*w)$, gdje je act jednak očekivanom argumentu eksponencijalne funkcije.

$$b4 = \begin{bmatrix} -2actdx^2w \\ -4actdxdyw \\ -2actdy^2w \\ 4actw \end{bmatrix}$$

Za definiranje ovih matrica u kodu prvo je potrebno izračunati sve potencije dx_pot i dy_pot koje se traže u sustavu linearnih jednadžbi i desnom stupcu. Nakon toga se definiraju matrice, odnosno vektori sa potencijama članova dx i dy za sve elemente matrice $A4$ i stupca $b4$. Na isti način su definirani i konstantni faktori za sve članove matrice i stupca. Matrica $A4$ ovisi samo o relativnim pozicijama točaka dx_i i dy_i, te o težinskoj funkciji svake točke w_i. Kao četvrto rješenje, matrica daje optimalni pomak targeta $z0$.

Sljedeće se formira logaritamski target za sustav s četiri nepoznanice. Za to je potrebno nepotpuno normalizirane ulazne podatke zn/dxy podijeliti s rezidualnom skalom, te faktor amplitudne skale modela pomnožiti s rezidualnom skalom. Time su ulazni podaci prilagođeni optimalnom modelu po skali. Provedena je i inicijalizacija stupca $b4$.

```
double *lz4;
lz4 = mxCalloc(z_length, sizeof(double));
for (i = 0; i < z_length; i++)
        lz4[i] = log(zn[i] / dxy + DBL_EPSILON);

double *b4;
b4 = mxCalloc(4, sizeof(double));
double *dx_dy_pot_lz4;
dx_dy_pot_lz4 = mxCalloc(z_length, sizeof(double));
//nastavak na sljedecoj stranici
```

```
for (i = 0; i < 4; i++) {
        for (m = 0; m < z_length; m++)
                dx_dy_pot_lz4[m] = dxpot[m+z_length*potb4_x[i]] *
                                   dypot[m+z_length*potb4_y[i]] * lz4[m];
        for (m = 0; m < z_length; m++)
                b4[i] += w[m] * dx_dy_pot_lz4[m] * fakb4[i];
}
```

Nakon toga se nalaze rješenja proširenog linearnog sustava $Ci11$, $Ci12$, $Ci22$ i $z0$ rješavanjem jednadžbe $Cijz0 = A4^{-1}b4$. Prva tri argumenta vektora $Cijz0$ su novi koeficijenti inverzne matrice kovarijance $Ci11$, $Ci12$ i $Ci22$, a zadnji argument je optimalni vertikalni pomak $z0$. Argument eksponencijalne funkcije modela za zadane (x_i, y_i) parove ulaznih vrijednosti se računa na sljedeći način:

```
double *arg;
arg = mxCalloc(z_length, sizeof(double));
for (i = 0; i < z_length; i++)
        arg[i] = -(Cijz0[0] * dxpot[i+z_length*2] / 2 + Cijz0[1] *
                 dx[i] * dy[i] + Cijz0[2] * dypot[i+z_length*2] / 2);
```

Problem nepoznavanja stvarne amplitude skale i nepoznavanja normalizacijskog faktora Gaussa se rješava pronalaskom greške stvarnog argumenta u odnosu na empirijski argument koji je vertikalno pomaknut za $z0$. Pronalaskom greške $como$, moguće je izračunati sumarnu kvadratnu težinsku pogrešku modela u logaritamskoj domeni $logerr4$.

```
for (i = 0; i < z_length; i++)
        como[i] = lz4[i] - z0 - arg[i];
mxFree(arg);

double logerr4 = 0;
for (i = 0; i < z_length; i++)
        logerr4 += w[i] * pow(como[i],2);
```

Kako bi bilo moguće odrediti model iterativnim analitičkim postupkom determinanta matrice kovarijance C i njezine inverzne matrice Ci moraju biti pozitivne. Stoga je dovoljno odrediti determinantu inverzne matrice i provjeriti uvjet za nju.

```
double detC4i;
detC4i = (Ci11 * Ci22 - pow(Ci12,2));
//nastavak na sljedecoj stranici
```

```
if (detC4i <= 0)
        break;
```

Ako inverzna matrica Ci zadovoljava uvjet, od njezinih članova se jednostavnim računom može formirati matrica kovarijance C za optimalno rješenje. Nakon što je dobivena matrica C još se jednom provjerava uvjet da su njezin prvi i zadnji element pozitivni. Ako jesu, iz elemenata matrice mogu se odrediti parametri koreliranog modela $s1$, $s2$ i ro.

```
s1 = sqrt(C[0]);
s2 = sqrt(C[3]);
ro = (C[1] / s1) / s2;
```

Us slučaju da je ro veći od jedan rješenje nije regularno i ne može se izračunati realna determinanta matrice C pa se prekida daljnja iteracija. U daljnjem postupku računa rezidualni logaritamski target lz uklanjanjem optimalnog pomaka $z0$ iz $lz4$.

```
for (i = 0; i < z_length; i++)
        lz[i] = lz4[i] - z0;
```

Nakon toga započinje izračun novog centroida modela koji će dodatno smanjiti sumarnu težinsku kvadratnu pogrešku argumenta. Na početku se definira inicijalni centroid za koji je Ci optimalan i u vektor $Covi$ se spremaju vrijednosti elemenata matrice Ci.

```
double *cen_init;
cen_init = mxCalloc(2, sizeof(double));
cen_init[0] = x0[0];
cen_init[1] = y0[0];

double *Covi;
Covi = mxCalloc(3, sizeof(double));
Covi[0] = Ci11;
Covi[1] = Ci12;
Covi[2] = Ci22;
```

Sljedeće je potrebno naći optimalno rješenje parametara centroida Gaussa za prethodno izračunate Ci i lz. Ono se nalazi iterativnim postupkom u kojem se poziva *gauss_centroid_error* funkcija (detaljno opisana u 6. poglavlju) i koristeći formulu za Newtonovu metodu optimizacije provjerava dali je nađen minimum funkcije i time

zadovoljen uvjet za izlazak iz petlje. Korištena formula za Newtonovu metodu optimizacije glasi:

$$x_{k+1} = x_k - \gamma[f''(x_k)]^{-1} f'(x_k),$$

gdje su umjesto x_{k+1} i x_k korišteni *cen_opt* i *cen_init*, a $f''(x_k)$ i $f'(x_k)$ su Hessianova matrica i gradijent po $x0$ i $y0$. γ je korak koji služi za točniju aproksimaciju optimalnog rješenja i njegova vrijednost je u rasponu $0 < \gamma < 1$. U ovoj implementaciji najveći omjer točnosti i brzine izvođenja dobiven je kada γ ima vrijednost 0.8.

```
while (1) {
        double *centroid_error;
        centroid_error = mxCalloc(7, sizeof(double));
        gauss_centroid_error(cen_init, Covi, v, w, lz, z_length,
                             centroid_error);

        iter++;

        Dw_min = centroid_error[0];
        grad[0] = centroid_error[1];
        grad[1] = centroid_error[2];
        hessian[0][0] = centroid_error[3];
        hessian[0][1] = centroid_error[4];
        hessian[1][0] = centroid_error[5];
        hessian[1][1] = centroid_error[6];
        mxFree(centroid_error);

        double hessian_inv[2][2];
        double hessian_inv_numer[2][2];
        hessian_inv_numer[0][0] = hessian[1][1];
        hessian_inv_numer[0][1] = -hessian[1][0];
        hessian_inv_numer[1][0] = -hessian[0][1];
        hessian_inv_numer[1][1] = hessian[0][0];
        for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++)
                hessian_inv[i][j] = hessian_inv_numer[i][j] /
                                    (hessian[0][0] * hessian[1][1] -
                                     hessian[0][1] * hessian[1][0]);
        }

        cen_opt[0] = cen_init[0] - 0.8 * grad[0] * hessian_inv[0][0];
        cen_opt[1] = cen_init[1] - 0.8 * grad[1] * hessian_inv[1][1];
        //nastavak na sljedecoj stranici
```

```
        if((grad[0] < pow(10, -14)) && (grad[1] < pow(10, -14))) {
                break;
        }
        else {
                cen_init[0] = cen_opt[0];
                cen_init[1] = cen_opt[1];
        }
}
```

Nakon pronalaska optimalnog rješenja i zadovoljenja uvjeta za izlazak iz petlje, zabilježava se broj iteracija potrebnih za konvergenciju centroida. Nove optimalne vrijednosti pozicije centroida spremaju se u varijable $x0$ i $y0$. Tada su dostupni svi podaci i moguće je formirati novi model s optimalnim parametrima koji minimiziraju težinsku kvadratnu pogrešku argumenta u logaritamskoj domeni.

```
double *mode_poly;
mode_poly = mxCalloc(5, sizeof(double));
mode_poly[0] = s1;
mode_poly[1] = s2;
mode_poly[2] = ro;
mode_poly[3] = x0[0];
mode_poly[4] = y0[0];
```

Sljedeće se računa normalizirani Gaussov model s optimalno određenom matricom kovarijance i optimalnim centroidom za domenu definiranu ulaznim podacima (x_i, y_i). To je izvedeno pozivanjem funkcije *gauss2D_fun_sparse* sa sljedećim parametrima:

```
gauss2D_fun_sparse(mode_poly,v1,v2,0,z_length,z1ok);
```

Sa vrijednostima vraćenim iz pomoćne funkcije sada je moguće izračunati optimalnu amplitudnu skalu usporedbom modela i empirijskih podataka, uz dodatno skaliranje modela.

```
z1ok_mul = 0;
z1okz_mul = 0;
for (i = 0; i < z_length; i++) {
        z1ok_mul += z1ok[i] * z1ok[i];
        z1okz_mul += z1ok[i] * z2[i];
}
skala[i_cent_iter] = z1okz_mul / z1ok_mul;
```

Parametri izračunatog modela $mode\_poly$ popunjavaju i_cent_iter redak matrice

$mode$, koja je predstavljena kao izlaz glavne funkcije. Ponovno se računa razlika empirijskih podataka i rafiniranog modela na isti način kao za inicijalni model, ali sa vrijednostima iz novog modela.

```
for (i = 0; i < z_length; i++)
        raz[i] = z2[i] - z1ok[i] * skala[i_cent_iter];

raz2_sum = 0;
z2_sum = 0;
for (i = 0; i < z_length; i++) {
        raz2_sum += pow(raz[i], 2);
        z2_sum += pow(z2[i], 2);
}
err[i_cent_iter] = 10 * log10(raz2_sum / z2_sum);
```

Pohranjuje se i nova sumarna težinska kvadratna pogreška argumenta u izlaznu varijablu $wearg$ radi provjere konvergencije.

```
wearg[i_cent_iter] = Dw_min;
```

Time je završena iteracijska petlja i dovršen rekurzivni postupak poboljšanja modela. Sada je potrebno izdvojiti samo valjane modele u slučaju prijevremenog izlaska iz vanjske petlje konvergencije. To je ostvareno izbacivanjem praznih redaka i zadržavanjem samo valjanih modela za sve varijable koje idu u ret_array. Na kraju glavne funkcije preostaje samo spremiti maloprije definirane valjane modele u ret_array, po rasporedu definiranom u 3.1 poglavlju.

4. *gauss2D_fun_sparse* funkcija

Funkcija kao parametre redom sadrži vektor $mode$, stupce x i y, $mode\_typ$ koji određuje dali će model biti opisan pomoću koreliranog ($mod\_typ = 0$) ili nekoreliranog modela ($mod\_typ = 1$), x_length koji sadrži duljinu ulaznih stupaca, te izlazni vektor koji se koristi pomoću pokazivača arr. Sama funkcija koristi se za izračunavanje 2D Gaussove funkcije definirane parovima vrijednosti u stupcima x i y. Spomenuti parovi ne moraju obuhvaćati cijelu mrežu, već mogu biti definirani samo za dio funkcije unutar kojeg se provodi optimizacija modela. U ovoj implementaciji korišten je opis preko parametara koreliranog modela.

Prvi cilj je izračunati matricu kovarijance i njezin inverz. Za to su potrebne varijable $s1$, $s2$, rho, $x0$ i $y0$ koje su prepisane iz ulaznog vektora $mode$ i redom označavaju:

s1: standardna devijacija po prvoj osi,

s2: standardna devijacija po drugoj osi,

rho: faktor normalizirane kross-korelacije prve i druge osi,

x0: sredina po x-osi,

y0: sredina po y-osi.

Koristeći te varijable poziva se funkcija *corr2decorr* koja preračunava korelirani u dekorelirani model, ona je detaljnije opisana u sljedećem poglavlju.

Nakon preračuna u dekorelirani model, pomoću pokazivača se čita vektor koji sadrži izlaze iz *corr2decorr* i te se vrijednosti onda spremaju u trenutno korištenoj funkciji. Koristeći standardne devijacije moguće je izračunati normalizacijski faktor fak:

```
double fak = 1 / (2 * M_PI * l1 * l2);
```

Sljedeće se definira vektor v koji sadrži ulazne stupce x i y kojima su oduzete sredine po x, odnosno y osi. Konačno, računa se Gaussova funkcija dvije varijable kao umnožak normizacijskog faktora i eksponencijalne funkcije s argumentom koji sadrži umnožak v vektora, $Cinv$ matrice i transponiranog v vektora.

```
double *mult;
mult = mxCalloc(x_length*2, sizeof(double));
for (i = 0; i < x_length; i++) {
        for (j = 0; j < 2; j++) {
                for (k = 0; k < 2; k++)
                        mult[i+x_length*j] += v[i+x_length*k] *
                                              Cinv[k][j];
        }
}

double *v_mult;
v_mult = mxCalloc(x_length*5, sizeof(double));
for (i = 0; i < x_length; i++) {
        for (j = 0; j < 2; j++)
                v_mult[i+x_length*j] = mult[i+x_length*j] *
                                       v[i+x_length*j];
}

mxFree(mult);
mxFree(v);

double *v_mult_trans;
v_mult_trans = mxCalloc(2*x_length, sizeof(double));
for (i = 0; i < x_length; i++) {
        for (j = 0; j < 2; j++)
                v_mult_trans[j+2*i] = v_mult[i+x_length*j];
}

mxFree(v_mult);

double *v_sum;
v_sum = mxCalloc(x_length, sizeof(double));
for (i = 0; i < x_length; i++)
        v_sum[i] = 0;
for (i = 0; i < x_length; i++) {
        for (j = 0; j < 2; j++)
                v_sum[i] += -0.5 * v_mult_trans[j+2*i];
}

mxFree(v_mult_trans);
//nastavak na sljedecoj stranici
```

```
for (i = 0; i < x_length; i++) {
        arr[i] = fak * exp(v_sum[i]);
}
```

5. *corr2decorr* funkcija

Ulazi u funkciju su $s1$, $s2$, rho, te $debu$ koji se u ovoj implementaciji ne koristi i postavljen je u nulu, a kao izlaz se koristi pokazivač arr. U vektor arr se kao izlaz funkcije nakon izračuna modela spremaju:

l1: standardna devijacija glavne dekorelirane osi,

l2: standardna devijacija pomoćne dekorelirane osi,

te: kut zakreta koordinatnog sustava (theta), uvijek u rasponu od $-\pi/2$ do $\pi/2$,

rot_m: rotacijska matrica za kut te,

C: matrica kovarijance polaznog modela izračunata na osnovu parametara dekoreliranog modela,

Cinv: inverzna matrica kovarijance polaznog modela izračunata na osnovu parametara dekoreliranog modela.

Nakon izračuna kuta zakreta te, potrebno je definirati matricu rotacije rot_m s obzirom na taj kut. Matrica je određena kao:

```
double rot_m[2][2] = {{cos(te), -sin(te)}, {sin(te), cos(te)}};
```

Standardne devijacije dekoreliranog modela $l1$ i $l2$ izračunate su koristeći izraze za sumu i pozitivnu razliku varijanci:

```
double ra = sqrt(fabs(pow(s1,4) +
            pow(s1,2)*pow(s2,2)*(4*pow(rho,2)-2) + pow(s2,4)));

double su = pow(s1,2) + pow(s2,2);

double l1 = sqrt((su + ra) / 2);
double l2 = sqrt((su - ra) / 2);
```

Matrica kovarijance C dobivena je pomoću direktnog analitičkog izraza na osnovi parametara dekoreliranog modela:

```
double C[2][2];
C[0][0] = pow(cos(te),2) * pow(l1,2) + pow(sin(te),2) * pow(l2,2);
C[0][1] = cos(te) * sin(te) * (pow(l1,2) - pow(l2,2));
C[1][0] = cos(te) * sin(te) * (pow(l1,2) - pow(l2,2));
C[1][1] = pow(sin(te),2) * pow(l1,2) + pow(cos(te),2) * pow(l2,2);
```

Iz matrice kovarijance je tada jednostavno izračunati inverznu matricu kovarijance $Cinv$.

6. *gauss_centroid_error* funkcija

Funkcija služi za računanje težinske kvadratne pogreške argumenta 2D Gaussove funkcije za zadane ulazne parametre. Parametri funkcije su središte Gaussove funkcije Cen, elementi inverzne matrice kovarijance $Covi$, v koji sadrži (x_i, y_i) parove za koje je zadana očekivana vrijednost argumenta u lz, pozitivna težina svakog uzorka w, očekivana negativna vrijednost argumenta eksponencijalne funkcije na zadanim pozicijama lzi, te duljinu stupaca v_length. Zadnji parametar je arr koji pomoću pokazivača vraća izlazne vrijednosti u glavnu funkciju. U arr se spremaju:

D: težinska kvadratna pogreška modela prema zadanim podacima,

dD: parcijalne derivacije D po $x0$ i $y0$,

H: Hessianova matrica s drugim derivacijama.

Na početku su izdvojene pozicije centroida $(x0, y0)$ iz ulaza Cen. Koristeći te vrijednosti računaju se dx i dy u odnosu na trenutačnu točku. dx je prvi stupac ulaza v umanjen za $x0$, dok je dy drugi stupac umanjen za $y0$. Iz parametra $Covi$ izvađeni su članovi i spremljeni u $Ci11$, $Ci12$ i $Ci22$ varijable. Koristeći spremljene parametre moguće je izračunati argument eksponencijalne funkcije za model arg po uzoru na glavnu funkciju. Kako bi se mogla izračunati težinska kvadratna pogreška za argument eksponencijalne funkcije D koja se vraća kao izlaz iz funkcije, prvo je potrebno odrediti težinsku razliku zadanih argumenta i argumenta izračunatih prema modelu.

```
double *di;
di = mxCalloc(v_length, sizeof(double));
for(i = 0; i < v_length; i++)
        di[i] = w[i] * (lzi[i] - arg[i]);

double D = 0;
for(i = 0; i < v_length; i++)
        D += w[i] * pow((lzi[i] - arg[i]),2);
```

Pomoćni izrazi potrebni za parcijalne derivacije i Hessian se računaju kao:

```
double *derxi;
double *deryi;
derxi = mxCalloc(v_length, sizeof(double));
deryi = mxCalloc(v_length, sizeof(double));
for(i = 0; i < v_length; i++) {
        derxi[i] = (Ci11 * dx[i] + Ci12 * dy[i]);
        deryi[i] = (Ci12 * dx[i] + Ci22 * dy[i]);
}
```

Parcijalne derivacije ukupne pogreške po $x0$ i $y0$ za trenutačni model se dobivaju množenjeme maloprije izračunatih pomoćnih izraza i pojedinačne pogreške svake točke uz skaliranje sa -2.

```
double dD[2] = {0, 0};
for(i = 0; i < v_length; i++) {
        dD[0] += -2 * derxi[i] * di[i];
        dD[1] += -2 * deryi[i] * di[i];
}
```

Sljedeće se definira sumarna težinska razlika sum_di kao suma svih vrijednosti iz vektora di. Vektor $scomo$ se dobiva množenjem sum_di sa članovima inverzne matrice kovarijance. Napokon, članovi Hessian matrice se računaju kao:

```
double H1[3] = {0, 0, 0};
double *w_derxi;
double *w_deryi;
w_derxi = mxCalloc(v_length, sizeof(double));
w_deryi = mxCalloc(v_length, sizeof(double));
for(i = 0; i < v_length; i++) {
        w_derxi[i] = w[i] * derxi[i];
        w_deryi[i] = w[i] * deryi[i];
}
for(i = 0; i < v_length; i++) {
        H1[0] += w_derxi[i] * derxi[i];
        H1[1] += w_derxi[i] * deryi[i];
        H1[2] += w_deryi[i] * deryi[i];
}
H1[0] = 2 * (H1[0] + scomo[0]);
H1[1] = 2 * (H1[1] + scomo[1]);
H1[2] = 2 * (H1[2] + scomo[2]);
//nastavak na sljedecoj stranici
```

```
mxFree(derxi);
mxFree(deryi);
mxFree(w_derxi);
mxFree(w_deryi);

double H[2][2];
H[0][0] = H1[0];
H[0][1] = H1[1];
H[1][0] = H1[1];
H[1][1] = H1[2];
```

7. Funkcija za povezivanje MATLAB-a i C funkcija

Za pokretanje C programa u MATLAB-u kao ulazna funkcija se ne koristi *main*, nego funkcija *mexFunction*. Ovoj funkciji MATLAB prosljeđuje broj ulaznih $(nrhs)$ i izlaznih argumenata $(nlhs)$, kao i same argumente pomoću polja pokazivača $(*plhs$ i $*prhs)$. Unutar funkcije se prvo ispisuje broj ulaznih argumenata te se provjerava dali je broj izlaznih argumenata veći od ulaznih, jer u tom slučaju dolazi do greške. Uz to se i formira matrica veličine potrebne za izlazne argumente. Nakon toga su definirani ulazi i izlazi kao pokazivači, nazvani isto kao u glavnoj funkciji, te pokazuju na $prhs$, odnosno $plhs$ argumente. Sa svim definiranim parametrima, sada je moguće pozvati glavnu funkciju:

```
gauss2Dest_sparse_polyest2(z, v, rub, dxy, cent_iter, debu, z_length,
                           ret_array);
```

8. Usporedba brzine izvođenja i točnosti MATLAB funkcije i C programa

8.1. Prevođenje i pokretanje C programa

Nakon što su funkcije napisane i glavna povezana sa *mexFunction*, potrebno ih je prevesti. Za to se u MATLAB-u koristi MinGW64 prevodioc, a program se prevodi naredbom mex, uz koju se navode glavna i sve pomoćne funkcije:

```
mex gauss2Dest_sparse_polyest2.c gauss2D_fun_sparse.c ...
corr2decorr.c decorr2corr.c gauss_centroid_error.c
```

Time je u trenutnom direktoriju stvorena MEX funkcija koja je nazvana isto kao prva funkcija navedena u naredbi. U slučaju da su svi parametri prethodno učitani u MATLAB i nazvani jednako kao u glavnoj funkciji, MEX funkcija pokreće se na sljedeći način:

```
ret_array = gauss2Dest_sparse_polyest2(z, v, rub, dxy, cent_iter, ...
            debu, z_length);
```

8.2. Usporedba brzine izvođenja i točnosti

Kako bi usporedba vremena izvođenja bila točna, oba su programa pozvana sa istim vrijednostima parametara. U odabranom primjeru duljina stupaca vektora prva tri parametara iznosi 344, te njihove sve vrijednosti nema smisla prikazivati zbog njihove duljine. Vrijednosti ostalih parametara su navedeni u tablici 8.1.

Tablica 8.1: Vrijednosti parametara

Naziv	Vrijednost
dxy	0.027777777777778
$cent\_iter$	2
$debu$	0
z_length	344

Vremena izvođenja programa mjerena su pomoću naredbi tic i toc. Prosječna vremena izvođenja programa iznose:

Tablica 8.2: Usporedba brzine izvođenja programa

Programski jezik	Vrijeme izvođenja (ms)
C	2.4408
MATLAB	45.7781

Što se tiče točnosti, između dobivenih rezultata postoji minimalna razlika zbog nekih različitih tipova podataka i većeg broja iteracija u C programu. U matrici $mode$ u prva dva retka, tj. modela, nema nikakve razlike, a u zadnja dva retka maksimalna razlika je na devetu decimalu. Za faktor optimalne skale $skala$ maksimalna razlika je na šestu decimalu, a kod vektora ukupne pogreške modela u odnosu na empirijske podatke err na jedanaestu decimalu. Kod ukupne težinske kvadratne pogreške argumenta eksponencijalne funkcije $wearg$ najveća razlika je na sedmu decimalu. $iter\_detC$ je isti, a broj iteracija za konvergenciju centroida $iter\_cent$ je veći u C implementaciji, kao što je već spomenuto.

9. Zaključak

Implementacija zadanog primjera obrade dvodimenzionalne slike korištenjem C jezika potvrđuje se kao brže i bolje rješenje od već realizirane implementacije u MATLAB programskom okruženju. Izlazni rezultati su gotovo identične točnosti kao traženi, te su dobiveni otprilike 20 puta većom brzinom nego u originalnoj implementaciji. Također, MATLAB se pokazuje kao dobar alat za integriranje i pokretanje C funkcija pomoću MEX datoteka.

LITERATURA

[1] *Create C Source MEX File*. https://www.mathworks.com/help/matlab/matlab_external/standalone-example.html?s_tid=mwa_osa_a.

[2] *fminunc*. https://www.mathworks.com/help/optim/ug/fminunc.html#d120e82422.

[3] Pascal Getreuer. *Writing MATLAB C/MEX Code*. UCLA. https://www.caam.rice.edu/~optimization/L1/optseminar/Getreuer-slides-cmex.pdf.

[4] Jan Šnajder. *Strojno učenje: 13. Procjena parametara*, 2017. Natuknice s predavanja.

[5] Tomislav Petković. *Kratke upute za korištenje MATLAB-a*, 2005. Zagreb.

ANALIZA I MODELIRANJE OBJEKATA U SLICI 2D GAUSSOVIM PROFILOM

Sažetak

U radu je realizirana implementacija obrade dvodimenzionalne slike u kojoj se identificiraju višestruki objekti njihovim modeliranjem pomoću multivarijantne Gaussove distribucije. Zadatak je rješen koristeći C jezik i pokreće se u MATLAB programskom okruženju. Ispravnost dobivenih rješenja je provjerena usporedbom sa originalnom implementacijom u MATLAB-u.

Ključne riječi: multivarijantna Gaussova razdioba, C, MEX, MATLAB

ANALYSIS AND MODELING OF IMAGE OBJECTS BASED ON 2D GAUSSIAN PROFILE FITTING

Abstract

This paper describes implementation of two-dimensional image processing in which multiple objects are identified by their modeling using multivariate Gaussian distribution. The problem is solved using C programming language and runs in the MATLAB programming environment. The solution's validity is verified by comparison with the original implementation in MATLAB.

Keywords: multivariate Gaussian distribution, C, MEX, MATLAB