4. feladat

Kincskeresés vektorosan

Ismerősünk ismét segítségünket kérte. Nagyapja naplójában talált egy feljegyzést arról, hogy hol van egy kincs elrejtve: "Lépj előre x_0 lépést, jobbra fordulva x_1 lépést, majd balra fordulva x_2 lépést, és ezt ismételd!" Azt sejti, hogy honnan kell indulni, csakhogy nagyapja a hely eléréséhez szükséges lépések számából álló x vektort beszorozta egy $\mathbf{A}$ mátrixxal és az eredményül kapott b vektort írta le. Szerencsére eszébe jutott, hogy nagyapjával matekot is játszottak régebben, és elővette "Játékos matek" feliratú dossziéját. Ebben két megfelelő méretű mátrixot is talált, ezek egyike lehet, amit nagyapja használt.

Készítsünk C programot, mely megvalósítja a Gauss-Jordan-elimináció algoritmusát, vagyis megoldja az $\mathbf{A} \cdot x = b$ lineáris egyenletrendszert! Itt $\mathbf{A}$ egy $N \times N$ méretű adott mátrix, x és b pedig N elemű oszlopvektorok.

Lépésenként bővítsük a programot az alábbi pontok szerint, és a beadott fájl nevében utaljunk rá, hogy melyik szinten működik, mégpedig main-a.c, main-b.c stb szerint. A főbb lépésekre, amelyekre az adott feladatrészhez szükség van, írjunk külön függvényeket: beolvasás, egy sor osztása egy számmal, egy sor számszorosának hozzáadása másik sorhoz, egy sor normálása, abszolút értékbeli maximumkeresés egy sorban, ill. egy oszloprészben, ill. egy almátrixban, két sor felcserélése, stb. (ahogy azt az előadás anyaga is sugallja).

A program a következő parancsargumentumokat használja ebben a sorrendben:

$\mathrm{A\_mátrix\_fájlja}\ \ \mathrm{b\_vektor\_fájlja}\ \ N$

A kincs és a pontszám eléréséhez a programnak az egyenletet teljesítő megoldást kell adnia az alábbi mátrixokkal és vektorokkal. Az első hármat csak tesztelésre szánjuk, azok egy egyenletrendszert teljesítő $\mathbf{A}$, x, b tömbök. A másik három a feljegyzések közt talált mátrix és vektor, amikhez keressük a megoldásvektort. Természetesen lehet saját kis mátrixokkal is ellenőrizni a program helyes működését..

A3x3.dat, x3x1.dat, b3x1.dat, Beauty6x6.dat, Beast6x6.dat, b6x1.dat

a) Gauss-Jordan-elimináció pivotálás nélkül

Írjuk meg a Gauss-Jordan-elimináció algoritmusát pivotálás nélkül és próbáljuk ki az algoritmust! A Beauty6x6.dat használatánál, ill kisebb mátrixok esetén általában megbízható eredményt kapunk. Ezt lehet ellenőrizni úgy, hogy miután kiszámoltuk az x megoldásvektort, az $\mathbf{A} \cdot x$ szorzás elvégzésével vissza kell kapnunk a b vektort (ezt az egyszerű számolást pl pythonban is meg lehet írni, de ezt nem kell beadni).

Tanulságos viszont, hogy vannak furcsán viselkedő, ú.n. rosszul kondicionált mátrixok, ilyen a Beast6x6.dat-ban tárolt is. Ezek esetében az egymástól kicsit eltérő (de helyes) programok egymástól lényegesen eltérő x eredményvektort adhatnak. Ugyanakkor mindegyik eredményre az $\mathbf{A} \cdot x$ szorzat nagy pontossággal visszaadja a b vektort. Összehasonlításul

megadjuk, hogy egy extra pontos eljárás az első három elemre azt adta ki, hogy $(60, 50, 40)$. Ha a Beauty mátrixxal a program jól működik, csak a Beasttel pontatlan, akkor nem szükséges megpróbálni ezen javítani. (A)

b) A LAPACK csomag használata

Ismerkedjünk meg a LAPACK numerikus lineáris algebra könyvtár használatával! Végezze el a program az előbbi számolást a LAPACK csomag használatával is. A kétféle számolás egymás után lefuthat, hogy ne kelljen külön programváltozatot írni. Az egyenletrendszer megoldásához javasolt a dgesv függvény használata. Sorfolytonosan tárolt mátrix esetében a szokásos egyszerű tárolás esetén így tudjuk meghívni:

> info = LAPACKE_dgesv(LAPACK_ROW_MAJOR, n, 1, mx, n, ipiv, v, 1);

ahol mx a mátrix pointere, v pedig a vektoré, ami kezdetben a b vektort kell tartalmazza, és a függvény lefutása után a megoldás vektort kapjuk meg benne. n a mátrix és vektor közös mérete. ipiv egy n elemű int tömb pointere kell legyen (a lefutás után kiegészítő információkat tartalmaz)

A LAPACK csomagot mindenki telepítheti saját gépére, ill. lehet használni a k8plex rendszerben is. A k8plexen való használat esetén ahhoz, hogy elérjük a programból a kívánt függvényt a programban a

> #include <lapacke.h>

sorra van szükség, a fordításkor pedig

> gcc -Wall main.c -llapacke

módon kell a csomagot megadni. Ahogy a fenti parancssor mutatja, itt a -llapacke opciónak a legvégén kell lennie! (A)

c) Részleges pivotálás

Bővítsük a programunkat részleges pivotálással (azaz a Gauss-eliminációs lépések előtt végezzük el a sorok normálását a maximális abszolút értékű elemükkel, másrészt a lineárkombinációkhoz válasszunk az oszlopokból legmegfelelőbb pivot elemet). Az eliminació során a programnak detektálnia kell, ha a mátrix numerikusan szinguláris, azaz a pivotelemnek választható elemek abszolút értékben kisebbek mint 10^{-8}! Írja ki, hogy melyik oszlopban ütközött problémába és mekkora a legnagyobb pivotelem! (T)

d) Skálázás

Szorgalmi feladat: Gyártsunk nagyméretű, véletlen elemű mátrixokat és vektort, és hasonlítsuk össze a saját és a LAPACK változat sebességét, illetve skálázását! Skálázás alatt azt értjük, hogy hogyan függ a futási idő az lineáris egyenletek N számától. Ábrázoljuk a futási időt az N függvényében és próbáljunk hatványfüggvényt illeszteni az adatokra!

A programot úgy írjuk meg, hogy a bemeneti paramétereket kétféleképpen is tudja kezelni: a fentiekben már ismertetett módon, vagyis amikor két tömböt és ezek méretét adjuk neki át,

valamint úgy is, hogyha az első parancsargumentumban "-r" karaktersort találja, akkor csak két argumentumot várjon, és a második argumentumban megadott méretben hozzon létre egy véletlen mátrixot és vektort. A skálázás vizsgálata esetén legyen a beolvasott N a legnagyobb vizsgált méret.

e) Teljes pivotálás

Szorgalmi feladat: írjuk meg a Gauss-Jordan eliminációt teljes pivotálással is! Hasonlítsuk össze a Beauty és a Beast matrixok esetén a részleges és a teljes pivotálással kapott eredményeket!