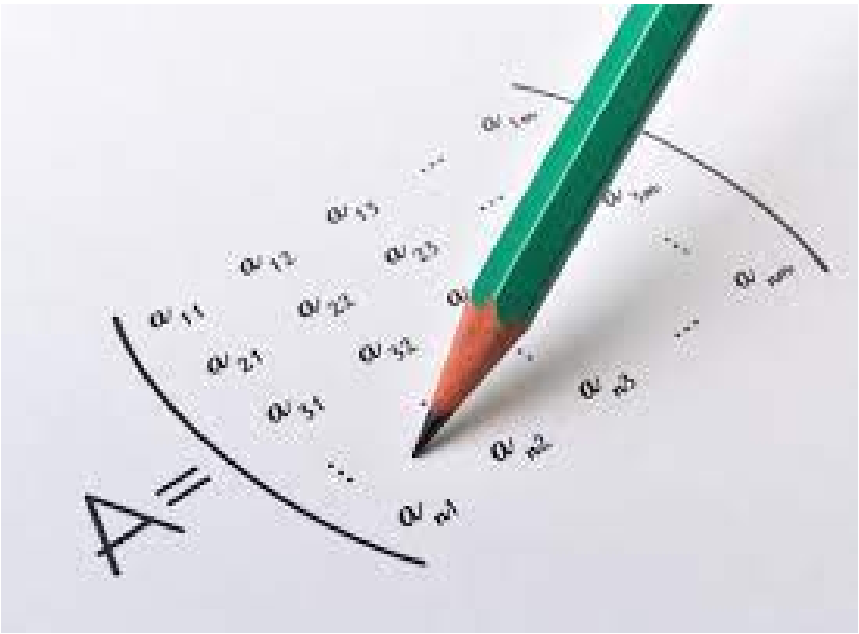
PRÀCTICA 1 D'INTRODUCCIÓ A LA COMPUTACIÓ CIENTÍFICA (ICC): Àlgebra lineal numèrica



Oscar De Caralt Roy
Enginyeria Informàtica UB:
2n de Carrera
Curs 2021-2022

Índex:

- **Introducció i objectius…… 2**
- **Treball fet amb explicacions…… 2-8**
- **Problemes…… 8**
- **Conclusions…… 9**

Introducció i objectius:

En aquesta pràctica volem implementar en C les rutines bàsiques per resoldre sistemes lineals usant el mètode de Gauss (sense o amb pivotatge) i algunes aplicacions. Donat n ≥ 2, considerem el sistema lineal Ax = b on x, b ∈ R n i A ∈ R n×R n és una matriu de rang n.

Treball fet:

int resoltrisup(int n, double **A, double *b, double *x, double tol):

$$x_{n-1} = b_{n-1}/a_{n-1,n-1}, \qquad x_i = \left(b_i - \sum_{k=i+1}^{n-1} a_{ik}\, x_k\right)/a_{ii} \quad i = n-2, \ldots, 1, 0.$$

2na part de l'algorisme

1ra part de l'algorisme

Rebrà com a paràmetres: la dimensió n, la matriu A, el vector b, el vector x i la tolerància acceptada tol i resoldrà una matriu A triangular superior. Declarem 2 integers, 1 per recórrer les columnes (cont1 en el document cont1 = i) i un altre per recórrer les files (cont2 en el document cont2 = k). Primer de tot, comprovarem que les condicions inicials es compleixen, si no es compleixen retornarem 1 i s'acabarà el programa, en canvi si es compleixen seguirem endavant. Fem un primer for on inicialitzem el cont1 amb valor n-2 i per cada volta restarem 1 al seu valor. Aquest for seguirà utilitzant-se fins que arribem a un valor de cont1 negatiu. Dins del for primer declarem que (x[cont1] = b[cont1]) i després utilitzem fabsf per comprovar que | A[cont1][cont1] | < tolerància donada, si es compleix passarem al pròxim for. Declarem un segon for on inicialitzarem el valor de cont2 amb cont1 -1. Aquest for seguirà fins que cont2 sigui menor o igual a n-1, afegint 1 al valor de cont2 a cada volta del bucle. Apliquem la primera condició de substitució endarrera (x[cont1] -= (A[cont1][cont2] * x[cont2])) i fora d'aquest 2n for però dins del primer, calculem la segona part de la substitució endarrera (x[cont1] /= A[cont1][cont1])

void resoltriinf(int n, double **L, double *x, double *b):

Rebrà com a paràmetres: la dimensió n, la matriu L (A trasposada), el vector b i el vector x i resoldrà una matriu triangular inferior. La idea és seguir la mateixa dinàmica que en resoltrisup però aquest cop variant una mica com recorrem els fors i emprant la substitució cap endavant. Jo he utilitzat aquest algorisme:

$$x_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij} x_j \right).$$

void prodMatVec (int m, int n, double **A, double *x, double *y):

Rebrà com a paràmetres: la dimensió de la matriu n, l'amplada del vector m, la matriu A, el vector y i el vector x i resoldrà el producte d'una matriu per un vector. Declarem 2 integers, 1 per recórrer les columnes (i) i un altre per recórrer les files (j). I anirem resolent pas a pas com faríem en paper (exactament igual). Primer, agafem el sumatori de les multiplicacions de la component i de la primera fila de la nostra matriu per l'element i del vector amb el que multipliquem, i el resultat que anem acumulant es va guardant en la primera posició del vector resultat (y en aquest cas). I repetim aquest procés n vegades guardant a la posició i del vector y el resultat que es va acumulant, corresponent a la multiplicació de cada component de la fila de la matriu per la posició corresponent de cada component del vector(= fila de la matriu) (tantes vegades com files tingui la matriu).

main_trisup:

Primer declarem les variables que utilitzarem i reservem espai per als vectors i per les matrius (fent ús de nom_vector = (tipus_de_dada_que_guarda_el_vector*) malloc(n*sizeof(tipus_de_dada_que_guarda_el_vector) i fent ús d'un for on la variable (i per exemple) creix a cada loop i on assignem Matriu[i] = (tipus_de_dada_que_guarda_la_matriu*)malloc(n*sizeof(tipus_de_dada_que_guard a_la_matriu));. Després introduir les dades de la matriu A amb un doble for i demanant a l'usuari que entri les dades per teclat. El mateix amb el el vector b però

amb només un for on demanem a l'usuari que entri les dades per teclat i de la tolerància.

Per entrar per teclat utilitzarem scanf("%le", &El_que_volguem_introduir_per_teclat); %le ens permetrà introduir el valor que introduim com a punt flotant llarg amb exponent.
Cridem al mètode resoltrisup que modificarà la matriu A per a que posteriorment el mètode prodMatVec modifiqui al vector x i calculem i imprimim el vector solució i la norma.

Després per a poder comprovar si funcionava el mètode resoltriinf hem transposat la matriu (amb el mètode transposar_matriu) que ens han introduit anteriorment (per així passar d'una triangular superior a una triangular inferior) i hem cridat a la funció resoltriinf amb la nova matriu. // aquesta part no acabava de funcionar, em donava code segmentation error probablement per un error en la funció resoltriinf així que l'he posat entre comentaris ja que no era part obligatoria i així tot funciona al 100% bé però aquella part també està ben feta (la del main, la de resoltriinf no del tot).
Per aquesta part, hem fet una funció extra anomenada transposar_matriu que intercanvia files per columnes.

Finalment, hem alliberat l'espai de la nostra memòria que havíem demanat anteriorment.

float prod_esc (int dim, float* x, float* y):

Literalment, el rescatem del lab2 on ja ens el donaven fet, però igualment l'explicaré. Rebrà com a paràmetres la dim dels 2 vectors (és la mateixa perquè sinó no podríem fer el producte escalar d'aquests), un vector de floats x i un altre vector de floats y. Declarem dins la funció una variable float prod (que farà de comptador), la inicialitzem i una altra variable (i) per a que recorri els 2 vectors. Agafarà l'element i del primer vector, el multiplicarà per l'element i del segon vector i l'afegirà el resultat a prod. Farà això fins que haguem recorregut tot el vector i després retornarà prod.

int gauss(int n, double **A, double *b, double tol):

Rebrà com a paràmetres: la dimensió n, la matriu A, el vector b i la tolerància acceptada tol. El seu objectiu és aconseguir posar 0's de manera escalada per a deixar un sistema triangular superior i així poder després cridar a la funció resoltrisup.

Fem 3 fors, en el primer bucle for iniciem cont2 amb valor 0 i a cada volta del bucle, anem augmentant el seu valor en 1, sortirem del bucle en el moment en que cont2 sigui igual o superior a n-1. En el segon for iniciem i amb valor cont2+1 i a cada volta del bucle, anem augmentant el seu valor en 1, sortirem del bucle en el moment en que i sigui igual o superior a n. En aquest bucle emprem fabs per a comparar si A[cont2][cont2]) >= tol, si és així calculem el multiplicador que guardarem en una variable temp per a després un cop finalitzat el tercer i el segon bucle, utilitzar-la. En el tercer for iniciem cont3 amb valor cont2 i a cada volta del bucle, anem augmentant el seu valor en 1, sortirem del bucle en el moment en que cont3 sigui més gran o igual a n. En aquest tercer bucle, per actualitzar el valor de la fila de la matriu en la qual ens trobem fem la resta de la fila en la que estem menys "multiplicador" vegades la fila que utilitzem com a referència i al sortir d'aquest bucle fem el mateix però per actualitzar els valors del vector.
Seguim endavant i ara com ens ha quedat un sistema triangular superior, hem de cridar a resoltrisup per a què ens el resolgui i posteriorment retornem el que retorni el sistema triangular superior.

main_gauss:

Primer declarem les variables que utilitzarem i reservem espai per als vectors i per les matrius (fent ús de nom_vector = (tipus_de_dada_que_guarda_el_vector*) malloc(n*sizeof(tipus_de_dada_que_guarda_el_vector) i fent ús d'un for on la variable (i per exemple) creix a cada loop i on assignem Matriu[i] = (tipus_de_dada_que_guarda_la_matriu*)malloc(n*sizeof(tipus_de_dada_que_guard a_la_matriu));. Després introduir les dades de la matriu A amb un doble for i demanant a l'usuari que entri les dades per teclat. El mateix amb el vector b però

amb només un for on demanem a l'usuari que entri les dades per teclat i de la tolerància.

Per entrar per teclat utilitzarem scanf("%le", &El_que_volguem_introduir_per_teclat); %le ens permetrà introduir el valor que introduim com a punt flotant llarg amb exponent.
Cridem al mètode gauss que modificarà la matriu A per a que posteriorment el mètode prodMatVec (que utilitza la matriu A original (A2) perquè A ha estat modificada al fer la crida de Gauss) modifiqui al vector x i calculem i imprimim el vector solució i la norma.

Finalment, hem alliberat l'espai de la nostra memòria que havíem demanat anteriorment.

double generateACP(int n, double **A, double *b, double tol):

Rebrà com a paràmetres: la dimensió n, la matriu A, el vector b i la tolerància acceptada tol.
Aquesta funció té el mateix codi que el mètode gauss afegint només una condició, perquè el que es busca es generar una matriu triangular superior, però que en comptes de tenir 0 al triangle inferior, tingui els multiplicadors emprats a cada pas. Gauss ens genera una matriu amb 0's al triangle inferior, però si a mesura que avancem, en comptes de posar 0's(que sortirien de fila actual menys mult vegades la fila amb la que estem comparant) posem els mults, obtindrem la matriu acp. La condició és un cop acabat el tercer bucle però dins del segon posem que A[i][j] = multiplicador.

double checkLU(int n, double **a, double **acp):

Rebrà com a paràmetres: la dimensió n, la matriu original (a) i la matriu modificada en la funció generateACP anomenada acp. Aquest mètode el que farà és comprovar que A = LU.

Fem 3 fors, en el primer bucle for iniciem i amb valor 0 i a cada volta del bucle, anem augmentant el seu valor en 1, sortirem del bucle en el moment en que cont2 sigui igual o superior a n. En el segon for iniciem j amb valor 'i' i a cada volta del bucle, anem augmentant el seu valor en 1, sortirem del bucle en el moment en que i sigui igual o superior o igual a n. En aquest bucle iniciem la nostra variable temp que ens ajudarà més tard. En el tercer for iniciem k amb valor 0 i a cada volta del bucle, anem augmentant el seu valor en 1, sortirem del bucle en el moment en que k sigui més gran o igual a n. Dins d'aquest tercer bucle modificarem el valor de l'auxiliar per a que dóni un valor d'aux actualitzat segons LU.
Després recorrem la matriu a buscant el major valor i guardant-lo en la variable que finalment retornarem. Aquesta funció no m'acaba de funcionar però no veig el perquè...

int gausspiv(int n, double **A, double *v, double tol):

Rebrà com a paràmetres: la dimensió n, la matriu A, el vector v i la tolerància acceptada tol. La idea és la mateixa que amb Gauss però abans de començar a operar mirar si el primer element de la fila que utilitzem com a referència és l'element més gran d'aquella columna. Per això, el que fem és recorrer la matriu buscant la fila amb el major pivot, i després permutar gràcies a variables auxiliars . Un cop ho tens, és el mateix algorisme que per a gauss, primer calcular les multiplicitats (xfac en aquest cas) i resoldre com hem fet anteriorment.

He intentat fer aquest mètode de varies maneres diferents, provant a inicialitzar de moltes maneres diferents els fors i revisant els ifs però em segueix donant <<Violación de segmento (`core' generado)>> . Com no acabo de veure on està l'error no puc arreglar-lo. He fet prints i he fet de tot per tal de debuggejar però no m'acava de sortir.

main_gausspiv:

Primer declarem les variables que utilitzarem i reservem espai per als vectors i per les matrius (fent ús de nom_vector = (tipus_de_dada_que_guarda_el_vector*) malloc(n*sizeof(tipus_de_dada_que_guarda_el_vector) i fent ús d'un for on la variable (i per exemple) creix a cada loop i on assignem Matriu[i] = (tipus_de_dada_que_guarda_la_matriu*)malloc(n*sizeof(tipus_de_dada_que_guard a_la_matriu));. Després introduir les dades de la matriu A amb un doble for i demanant a l'usuari que entri les dades per teclat. El mateix amb el vector b però amb només un for on demanem a l'usuari que entri les dades per teclat i de la tolerància.

Per entrar per teclat utilitzarem scanf("%le", &El_que_volguem_introduir_per_teclat); %le ens permetrà introduir el valor que introduim com a punt flotant llarg amb exponent.
Cridem al mètode gausspiv que modificarà la matriu A per a que posteriorment el mètode prodMatVec modifiqui al vector x i calculem i imprimim el vector solució i la norma.

Finalment, hem alliberat l'espai de la nostra memòria que havíem demanat anteriorment.

Problemes:

Per a la primera data d'entrega no m'acabava de funcionar la virtual box on tenia ubuntu, per la qual cosa no podia compilar i tenia molts errors però per a aquesta segona data d'entrega he decidit, instalar un segon sistema operatiu de forma dual i així poder ser capaç de compilar i resoldre errors més fàcilment. Tot i així, no m'ha sortit el mètode CheckLU del tot i gausspiv em dóna <<Violación de segmento (`core' generado)>>... Com no acabo de veure on està l'error no puc arreglar-lo. He fet prints i he fet de tot per tal de debuggejar però no m'acaba de sortir així que ho he acabat deixant així.

Conclusions:

En part, he aconseguit els objectius de la pràctica, però no estic del tot satisfet amb el treball realitzat ja que no està 100% bé. Però em veig capaç d'encarar molt millor futures pràctiques i intentaré evitar que torni a passar per a la pròxima entrega. Aquesta pràctica m'ha ajudat a aprendre bastant millor els algorismes practicats a les classes de problemes, el llenguatge c i com programar amb el terminal de linux. A més, he pogut concloure que, per a matrius de grans dimensions és molt més eficaç programar els càlculs que haver de fer-los manualment.