ΣΤΟΙΧΕΙΑ:

Όνομα	Επίθετο	Α.Μ
Ένρι	Γκάτση	1115200900048
Gerald	Mema	1115200800108

ΟΔΗΓΙΕΣ ΜΕΤΑΓΛΩΤΙΣΗΣ για το κυρίως πρόγραμμα
1)make

ΟΔΗΓΙΕΣ ΜΕΤΑΓΛΩΤΙΣΗΣ για τα tests.
1)make

ΟΔΗΓΙΕΣ ΧΡΗΣΗΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ
Εκτέλεση : οπως αναφέρεται στην εκφώνηση
Επιπρόσθετα :
Καθάρισμα : make clean

Προσοχή τα d1 και d2 να είναι σωστά στην κλήση του προγράμματος διαφορετικά θα υπάρξει segmentation fault.

ΟΔΗΓΙΕΣ ΧΡΗΣΗΣ των tests.
Εκτέλεση : ./main
Επιπρόσθετα :
Καθάρισμα : make clean

ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Επίλυση συστημάτων μη γραμμικών εξισώσεων.

Η άσκηση αυτή επικεντρώνεται στο να λύνει ένα πρόβλημα ιδιοτιμών-ιδιοδιανυσμάτων με το κλασσικό (στάνταρ) τρόπο ή με το γενικευμένο. Η συνάρτηση που επιλύει το πρόβλημα αυτό βρίσκεται στο handler με όνομα solve(). Αναπτύσσονται μέσα σε αυτή τα ερωτήματα της άσκησης.

(ερώτημα 5)Αρχικά ελέγχουμε αν μηδενίζεται η ορίζουσα του πίνακα Sylvester για κάθε κρυμμένη μεταβλητή. Αυτό το κάνει η συνάρτηση calcDet η οποία είναι υλοποιημένη στην κλάση Sylvester.

(ερώτημα 1)Το επόμενο βήμα είναι το πρώτο ερώτημα δηλαδή ο υπολογισμός του δείκτη κατάστασης (κ) του μεγιστοβάθμιου πίνακα πολυωνύμου μητρώου (Md) ,το οποίο μας δείχνει ανάλογα με την τιμή του αν ο πίνακας αντιστρέφεται ή όχι. Την τιμή αυτή (Β) την καθορίζει ο χρήστης .Ο υπολογισμός του κ γίνεται από την συνάρτηση handlerCalcKappa η οποία καλεί την generateKappa η οποία με την σειρά της καλεί την calcKappa και από αυτήν παίρνουμε το τελικό αποτέλεσμα.

(ερώτημα 2)Αν η τιμή του κ είναι μικρότερη από το όριο που έχουμε δώσει τότε υλοποιούμε το δεύτερο ερώτημα στο οποίο φτιάχνουμε τον πίνακα Companion και λύνουμε το στάνταρ πρόβλημα ιδιοτιμών-ιδιοδιανυσμάτων. Το πρόβλημα αυτό το αναθέτουμε στη κλάση Companion όπου φτιάχνεται αυτός ο πίνακας και για τις περιπτώσεις που ο βαθμός της κρυμμένης μεταβλητής είναι ίσο με ένα αλλά και για μεγαλύτερο του ένα. Καλείται η συνάρτηση solveWithInvert. Οι συναρτήσεις του περιγράφονται λεπτομερώς παρακάτω.

(ερώτημα 3)Αν η τιμή του κ είναι μεγαλύτερη από το όριο τότε θα πάμε να λύσουμε το γενικευμένο πρόβλημα ιδιοτιμών-ιδιοδιανυσμάτων φτιάχνοντας τους πίνακες L0 και L1 . Το πρόβλημα αυτό το επιλύει η κλάση GeneralProblem . Και εδώ τα L0 και L1 φτιάχνονται διαφορετικά για διαφορετικές περιπτώσεις ανάλογα με τον βαθμό της κρυμμένης μεταβλητής. Οι συναρτήσεις της κλάσης αυτής περιγράφονται λεπτομερώς παρακάτω.

(ερώτημα 4)Πριν όμως πάμε στο γενικευμένο πρόβλημα (αν οδηγηθούμε σε αυτό) ,εφαρμόζουμε αλλαγή κρυμμένης μεταβλητής και ξανά υπολογίζουμε το κ μήπως και βρούμε πιο μικρό ώστε να αποφύγουμε όσο μπορούμε το ill-conditioned πρόβλημα. Την υλοποίηση της παραπάνω διαδικασίας την κάνει η κλάση Zed η οποία καλείται μέχρι 3 φορές. Την αντικατάσταση από z σε y την κάνουμε πριν εφαρμοστεί το ερώτημα 7 και μετά πάμε και ελέγχουμε για ρίζες με y και x στον αρχικό πίνακα συντελεστών.

(ερώτημα 7 & 8)Με όποιο από τους δύο τρόπους επίλυσης κληθεί το πρόγραμμα, οι ιδιοτιμές και τα ιδιοδιανύσματα που μας επιστρέφει η Eigen θα πρέπει να επαληθευτούν στα αρχικά πολυώνυμα M1 και M2 για να επιβεβαιώσουμε αν είναι ρίζες του συστήματος. Οι ρίζες επαληθεύονται για πραγματικές αλλά και μιγαδικές

τιμές.
* Και οι συμπληρωματικές συναρτήσεις που έχουμε προσθέσει στις κλάσεις από την 1η άσκηση αναφέρονται παρακάτω.

ΚΑΤΑΛΟΓΟΣ ΤΟΥ ΑΡΧΕΙΟΥ ΚΩΔΙΚΑ/ΕΠΙΚΕΦΑΛΙΔΩΝ ΚΑΙ ΠΛΗΡΗ ΠΕΡΙΓΡΑΦΗ ΤΟΥΣ

Kappa.h Kappa.cpp

generateKappa(**double** * data,**int** grammes , **int** B)
: Καλεί την συνάρτηση JacobiSVD για να βρει τις ιδιάζουσες τιμές.

data = μονοδιάστατος πίνακας που θέλω να υπολογίσω το κ
B = B όπως δώθηκε μετά το -solve ή 7 by default

calcKappa: Επεξεργάζεται κατάλληλα τις ιδιάζουσες τιμές και στην συνέχεια κάνει σ(max)/σ(min) υπολογίζει τον δείκτη κατάστασης κ.

Companion.h Companion.cpp

makeCaseInvertable1(**double** * M0 , **double** * M1)
:Δίνει στην Eigen το C στην μορφή -M0(αντίστροφο) * M1 για να βρούμε τις ιδιοτιμές.

makeCaseInvertableD(**int** vathmosKrymenhsMetavlhths, **double** ** listOfMd)
: Δίνει στην Eigen το C στην μορφή ενός πίνακα που υλοποιού οι παρακάτω συναρτήσεις, για να βρούμε τις ιδιοτιμές.

generateFirstRowOfCompanion:
Φτιάχνεται η πρώτη γραμμή του C.

GenerateRestCompanion:
Κάνουμε shift τις υπόλοιπες γραμμές εκτος από το τελευταίο block του πίνακα.

GenerateLastBlockOfCompanion(**int** vathmosKrymenhsMetavlhthts, **double** ** list)
:
Φτιάχνουμε την τελευταία γραμμή(block).

allocateC(**int** grammes,**int** sthles)
:
Δεσμεύεται ο κατάλληλος χώρος για το C (m*d).

convertC:
Τον μετατρέπουμε σε μονοδιάστατο.

CalcEigenvectorsAndEigenValues: Βρίσκουμε τις ιδιοτιμές και τα ιδιοδιανύσματα , τα οποία μας επιστρέφει η Eigen.

calcX(EigenSolver<MatrixXd> & es)
:Φαίρνω τις ιδιοτιμές στην κατάλληλη μορφή (1,χ ,χ^2,...) .

SolveWithInvert: καλεί την makeCaseInvertable1 όταν ο βαθμός της κρυμμένης μεταβλητής είναι 1 και την makeCaseInvertableD για μεγαλύτερη του 1.

GeneralProblem.h GeneralProblem.cpp

makeCaseGeneralProblem1(**double** *M0, **double** *M1)
:
Απλή περίπτωση όπου d=1 .Δίνουμε τα L0 L1

makeCaseGeneralProblemD:
Όταν το d>1 τότε καλούμε τις παρακάτω συναρτήσεις

generateFirstRowOfL0:
Φτιάχνουμε την πρώτη γραμμή του L0 (0 -1 0 0 ....<blocks>).

generateFirstRowOfL1:
Φτιάχνουμε την πρώτη γραμμή του L1 (1 0 0 0 ....<blocks>).

GenerateRestL0L1: Κάνω shift τα L0 L1 μια θέση δεξιά.

GenerateLastBlockOfL0:(**int** vathmosKrymenhsMetavlhthts, **double** **list)
Φτιάχνουμε την τελευταία γραμμή του L0 (ανά block).

GenerateLastBlockOfL1(**int** vathmosKrymenhsMetavlhthts, **double** **list):
Φτιάχνουμε την τελευταία γραμμή του L1 (ανά block).

allocateL0L1(**int** grammes1, **int** sthles):
Δεσμεύεται χώρος για τα L0 L1 (m*d).

convertL0L1:
Μετατρέπουμε το L0 & L1 σε μονοδιάστατους πίνακες.

CalcEigenvectorsAndEigenValues: Χρησιμοποιούμε την συνάρτηση GEP για να βρούμε τις ιδιοτιμών-ιδιοδιανύσματα

calcX(MatrixXd & lambda, MatrixXd & V):
Υπολογίζουμε τις ιδιοτιμών-ιδιοδιανύσματα και τα φαίρνουμε στην σωστή κανονικοποιημένη μορφή.

solveWithGeneralProblem:
Καλείται στον handler και μας διευθύνει με βάση τον βαθμό d να δούμε ποιες συναρτήσεις θα χρησιμοποιήσουμε

GEP(MatrixXd & A,MatrixXd & B,MatrixXd & V,MatrixXd & lamda):

τα lambda και V σε μεταβλητές της κλάσσης.

Zed.h Zed.cpp

make1D: Μετατροπή σε μονοδιάστατο

calc_z(**int** *tempa, **int** *tempb, **int** *Ti, **int** power):Υπολογίζει τις δυνάμες του τ2 + τ1*ι και τ4 + τ3*ι

calkKappa:
Βρίσκει το κ και επιστρέφει 0 σε περίπτωση επιτυχίας

z_is_hidden_variable(Pinakas_polywnymou_mhtrwwn<**int***> * ppm):
Εδώ έχουμε τον αλγόριθμο για αλλαγή μεταβλητής σε πίνακα.

change_hidden_variable(Pinakas_polywnymou_mhtrwwn<**int***> * , **double** B):

Αλλαγή μεταβλητής στον μεγιστοβάθμιο πίνακα πολυωνύμου μητρώων (Md)

Επιστρέφει 0 αν αντιστρέφεται , αλλιώς 1.

change_PinakaSyntelestwn(Pinakas_syntelestwn<**int**> * ps):
Αλλαγή μεταβλητής στον πίνακα συντελεστών ps.

* Στην κλάση Sylvester έχουμε υλοποιήσει την συνάρτηση calcDet για τον υπολογισμό της ορίζουσας.

* Στην κλάση Pinakas_Syntelestwn έχουμε υλοποιήσει τις παρακάτω συναρτήσεις

findMultiplicity(MatrixXd & V,MatrixXd & lambda, **int** pos , **int** printMultiplicity): Ελέγχουμε αν μια ιδιοτιμή είναι διπλή. Αν έχει μόνο πραγματικό μέρος τότε το εμφανίζουμε χωρίς να βρούμε το χ αλλιώς (αν έχει φανταστικό μέρος) βρίσκουμε το χ και το εφαρμόζουμε στον αρχικό πίνακα συντελεστών(ερώτημα 8).

calcWith(Eigen::MatrixXd lambda, Eigen::MatrixXd V , **int** k , **int** printMultiplicity , **char** hv): Εφαρμόζουμε το 7 ερώτημα .Παίρνουμε μία πιθανή λύση και βλέπουμε αν όντως είναι ρίζες κάνοντας επαλήθευση στον αρχικό πίνακα συντελεστών.

calc_roots_with_imagine(**double** * y , **double** * x):Παίρνει το αποτέλεσμα που επιστρέφει η imag_roots και ελέγχει αν είναι όντως ρίζες.

imag_roots(**double** * result,**double** * y,**double** * x): Ουσιαστικά κάνει ό,τι κάνει η calcWith αλλά εδώ έχουμε έχουμε και φανταστικό μέρος.

Result = το αποτέλεσμα που θα επιστρέψει η επαλήθευση για ρίζες.
y = η τιμή της κρυμμένης μεταβλητής για τη συγκεκριμένη ιδοτιμή
χ = αντίστοιχη τιμή του ιδοδιανυσματος

calcWithImag(**double** * temp,**double** * variable,**int** power):Βοηθητική συνάρτηση της imag_roots.

temp=η τιμή που θα πάρει μια ταυτότητα της μορφής (α+ωι)^ν

variable=η μεταβλητή με πραγματικό και φανταστικό μέρος.

power=η δύναμη που θα υψωθεί

εκτελεί το variable^power και αποθηκεύει το αποτέλεσμα στο temp.

Tests:

Στο φάκελο libTest μπορούμε να βρούμε τη βιβλιοθήκη των test.
Στο κατάλογο TestLibMatrix έχουμε τις κλάσεις που τεστάρουν τις συναρτήσεις που έχουμε προσθέσει στο libMatrix στη δεύτερη άσκηση

testPinakas_polywnymou_mhtrwwn (κληρονομει την Pinakas_polywnymou_mhtrwwn)

Περιέχει μια συνάρτηση που τεστάρει τις public συναρτήσεις που προστέθηκαν στη Pinakas_polywnymou_mhtrwwn

Μέθοδος : testPublicFunctions()

Δημιουργεί ένα πίνακα και του αναθέτει τιμες , μετά ελέγχω εαν οι συναρτήσεις

convertToDouble // κάνει τα int double και τα αποθηκεύει σε ένα 2D πίνακα στη κλάσση
make1D // κάνει τον 2D σε 1D

και μετά ελέγχει τις τιμές αυτών των πινάκων αν είναι αυτές που πρέπει.

TestPinakas_syntelestwn

Μέθοδος calcWith():

Ελέγχει τη συνάρτηση calcWith που υπολογίζει τη τιμή μιας συνάρτησης σε συγκεκριμένο λαμδα και V .

Δίνω τιμές απο πετυχημένη εκτέλεση του προγράμματος και ελέγχω αν λειτουργεί με βάση αυτές τις τιμές.

Αν γυρίσει κάτι != 0 θεωρείτε λάθος.

TestPolywnymo_mhtrwwn

Ελέγχει τη συνάρτηση MakeArrayWithDoubleArrays και τσεκάρει τους πίνακες που γυρνάει αν έχουν σωστές τιμές.

TestCompanion

μέθοδος testsolveWithInvert():

Ελέχγει αν ο companion δουλεύει για διάφορα d με αντιστρέψιμο Md

TestGeneralProblem

μέθοδος testSolveWithGeneralProblem():

Ελέχγει αν ο GeneralProblem δουλεύει για διάφορα d.

TestKappa

μέθοδος testGenerateKappa()

Ελέγχει για πίνακες με μεγάλα και μικρά κάπα αν γυρνάει τη κατάλληλη τιμή.

TestZed

μέθοδος testCalcKappa()

Ελέγχει αν η συνάρτηση CalcKappa του Zed γυρνάει σωστά αποτελέσματα για συγκεκριμένους πίνακες που ξέρω τι κάπα να περιμένω.