ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ **ΜΗΧΑΝΙΚΩΝ**
ΠΛΗΡΟΦΟΡΙΑΚΩΝ &
ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Διπλωματική Εργασία

Πειραματική αξιολόγηση παιγνιοθεωρητικών στρατηγικών αρχηγού

Εισηγητές: Τσούλος Σωτήριος

Κοσμολίαπτσης Νικόδημος

Επιβλέπων καθηγητής: Καπόρης Αλέξιος

Περίληψη

Η παρούσα εργασία ασχολήθηκε με το testing και την βελτίωση του αλγορίθμου που αφορά το πρόβλημα knapsack. Αρμοδιότητες που ορίστηκαν, ήταν η πραγματοποίηση testing στον κώδικα, που επιμελήθηκε ο υπεύθυνος καθηγητής κ. Αλέξης Καπόρης, σε πειραματικό στάδιο, η συλλογή και καταγραφή των δεδομένων που προκύπταν καθώς επίσης και η σύγκριση αυτών μεταξύ τους ώστε να οδηγηθούμε σε ορισμένα συμπεράσματα. Για την ολοκλήρωση του πειραματικού στάδιο έγιναν επτά πειράματα με γραμμένο κώδικα σε τρία διαφορετικά προγραμματιστικά περιβάλλοντα :

- MATLAB[1]
- MAPLE[2]
- C++ μέσω CLion[3]

Με το πέρας της πειραματικής διαδικασίας, άρχισε η συλλογή και ομαδοποίηση των αποτελεσμάτων, ανά πείραμα, σε πίνακες ευπρεπώς παρουσιασμένοι σε αρχείο Word, αφού πρώτιστα είχαν αποθηκευτεί σε αρχεία txt, έτσι ώστε η σύγκριση των αποτελεσμάτων να είναι ευκολότερη. Τέλος, στο παρών αρχείο Word έγινε η αναλυτική περιγραφή της πειραματικής διαδικασίας εξηγώντας, με τα κατάλληλα screenshots, όλα τα βήματα που εκτελέστηκαν για την εξόρυξη των δεδομένων που μας απασχολούσαν, από τα τρία προγράμματα και στα επτά πειράματα.

Θα πρέπει επίσης να αναφέρουμε πως η επικοινωνία μας με τον υπεύθυνο καθηγητή, για οποίες απορίες, δυσκολίες και προβλήματα προκύπταν καθώς επίσης και η ενημέρωση της προόδου της διπλωματικής εργασίας γινόταν μέσω του webmail του Πανεπιστημίου Αιγαίου.

Summary

The present thesis dealt with the testing and improvement of the algorithm concerning the knapsack problem. The defined responsibilities were the testing of the code (edited by the professor in charge Mr. Alexios Kaporis) in an experimental stage, the collection and recording of the resulting data as well as the comparison between them in order to reach certain conclusions. To complete the experimental stage, seven experiments were completed with code written in three different programming environments :

- MATLAB[1]
- MAPLE[2]
- C++ via CLion[3]

At the end of the experimental process, began the collection and grouping of the results, per experiment, in tables presented in a Word file, after they were first saved in txt files, so that the comparison of the results is easier,. Finally, in this Word file a detailed description of the experimental process was made, explaining, with the appropriate screenshots, all the steps performed to extract the data that concerned us, from the three programs and the seven experiments.

We should also mention that our communication with the professor in charge, if any questions, difficulties and problems arose, as well as the updating of the progress of the thesis, was done through the webmail service of the University of the Aegean.

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες σε όλους όσους με στήριξαν καθ' όλη τη διάρκεια της ολοκλήρωσης της διπλωματικής μου εργασίας. Θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες στον σύμβουλος και επιβλέπων καθηγητή κ. Αλέξιο Καπόρη ,για την καθοδήγηση, την ενθάρρυνση και την ατελείωτη υποστήριξή του. Οι γνώσεις και η τεχνογνωσία του ήταν ανεκτίμητες στη διαμόρφωση της έρευνάς μου και στην πραγματοποίησή της.

Θα ήθελα επίσης να εκφράσω τις ευχαριστίες μου στους φίλους και την οικογένειά μου που μου παρείχαν συναισθηματική υποστήριξη και ενθάρρυνση κατά τη διάρκεια της μακράς και απαιτητικής διαδρομής συγγραφής της διατριβής μου. Η ακλόνητη αγάπη και η υποστήριξή τους ήταν πηγή δύναμης και έμπνευσης για μένα.

Τέλος, θα ήθελα να εκφράσω τις ευχαριστίες μου σε όλους όσους συνέβαλαν άμεσα ή έμμεσα στην έρευνά μου. Η συνεισφορά τους με βοήθησε να ολοκληρώσω με επιτυχία αυτή τη δουλειά.

Περιεχόμενα

Κεφάλαιο 1. Εισαγωγή

Σκοπός της παρούσας εργασίας είναι η πρωτότυπη πειραματική μελέτη παιγνιοθεωρητικών στρατηγικών αρχηγού. Μέσα από την πραγματοποίηση εφτά πειραμάτων καταφέραμε τη συλλογή αρκετών δεδομένων ώστε αυτά να συγκριθούν και να οδηγήσουν σε ασφαλή συμπεράσματα.

Η μεθοδολογία εργασίας γινόταν κατόπιν συνεννόησης με τον υπεύθυνο καθηγητή κ. Αλέξη Καπόρη. Αρχικά, εγκαταστήσαμε στους η/υ μας τα τρία αυτά προγράμματα και από ένα remote pc αντιγράφαμε τον έτοιμο κώδικα του εκάστοτε πειράματος και για τα 3 αυτά προγράμματα με σκοπό να κάνουμε testing να πάρουμε αποτελέσματα και από τα τρία προγράμματα και τελικώς να γίνει σύγκριση των αποτελεσμάτων. Τα βήματα των πειραμάτων εξηγούνται αναλυτικά στο κύριο κείμενο της διπλωματικής εργασίας.

Παρακάτω, εκτείνεται το κύριο κείμενο της εργασίας, η οποία είναι δομημένη με τρόπο ώστε να παρουσιάζονται αναλυτικά όλα τα βήματα που ακολουθήθηκαν για την πραγματοποίηση μιας ολοκληρωμένης πειραματικής γραμμής σε κάθε ένα από τα πειράματα που εκτελέσθηκαν, με χρήση στιγμιότυπων οθόνης (screenshots), επίσης η παράθεση συμπερασμάτων. Τέλος, παρατίθενται όλες οι χρήσιμες εντολές και βιβλιοθήκες που χρησιμοποιήθηκαν καθώς και το τι πληροφορίες αντλήσαμε από αυτές.

Κεφάλαιο 2. Πείραμα 1

2.1.1 Περιγραφή διαδικασίας (MAPLE[2])

Αρχίζοντας τη διαδικασία του 1ου πειράματος μπαίναμε στο MAPLE[2] και δοκιμάζαμε πραγματικούς αριθμούς στο input “alpha” όπως φαίνεται στον παρακάτω ψευδοκώδικα.

```
Function Y(m, l)
  return (2 - m)*(2 - 3*m)/(4 - 3*l) + m^2/l
End Function

alpha = 0.2556794561
mu = 2 * alpha
lambda_mu = solve for lambda in Y(mu, lambda) = 0, with lambda ranging from 0 to 2 * mu
Lc = mu / lambda_mu + 4
dC = 2 * mu / lambda_mu + 4
Lf = (2/3 - mu) / (4/3 - lambda_mu) + 4
dC_f = 2 * (2/3 - mu) / (4/3 - lambda_mu) + 4
d = 100
n = 8
a = array of 8 values [0.1, 0.15, 0.18, 0.17, 0.26, 0.3, 0.37, 0.42]
ad = a * d
A = sum of elements in a
Ad = sum of elements in ad
Ad = round down Ad to the nearest integer
W = Ad
for each element in ad
  round down the element to the nearest integer
End For
lambda_mu_A = lambda_mu * A
lambda_mu_A_d = lambda_mu_A * d
lambda_mu_A_d = round down lambda_mu_A_d to the nearest integer
Function C(m, l, A)
  return A * (8 + Y(m, l))
End Function
```

με σκοπό να πλησιάσουμε όσο το δυνατόν περισσότερο σε ακαίραιο.

$$\alpha := 0.14107223$$
$$\mu := 0.28214446$$
$$[0.14107223, 0.28214446, 0.343589744077227]$$
$$Lc := 4.82116671077523$$
$$dC := 5.64233342155046$$
$$Lf := 4.38850689297115$$
$$dC\_f := 4.77701378594229$$
$$a := [0.1, 0.15, 0.18, 0.17, 0.26, 0.3, 0.37, 0.42]$$
$$ad := [10.0, 15.00, 18.00, 17.00, 26.00, 30.0, 37.00, 42.00]$$
$$A := 1.95$$
$$Ad := 195.00$$
$$Ad := 195$$
$$0.670000000950593$$
$$67.0000000950593$$
$$67$$
$$C := (m, l, A) \mapsto A\,(8 + Y(m, l))$$
$$1.95$$

Όταν γινόταν αυτό είχαμε έτοιμο το output με τα δεδομένα a – dc – L της πρώτης γραμμής των αποτελεσμάτων μας όπως φαίνονται παρακάτω:

$$\alpha := 0.14107223$$
$$\mu := 0.28214446$$
$$[0.14107223, 0.28214446, 0.343589744077227]$$
$$Lc := 4.82116671077523$$
$$dC := 5.64233342155046$$
$$Lf := 4.38850689297115$$
$$dC\_f := 4.77701378594229$$
$$a := [0.1, 0.15, 0.18, 0.17, 0.26, 0.3, 0.37, 0.42]$$
$$ad := [10.0, 15.00, 18.00, 17.00, 26.00, 30.0, 37.00, 42.00]$$
$$A := 1.95$$
$$Ad := 195.00$$
$$Ad := 195$$

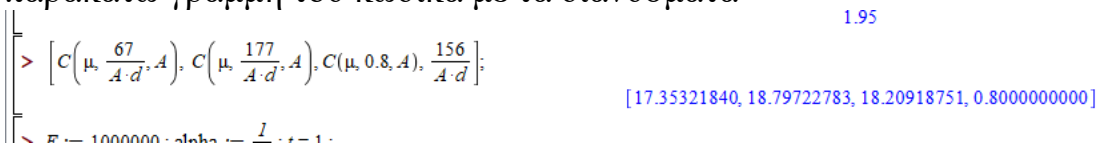
Για την ολοκλήρωση της πρώτης γραμμής υπολογίζαμε το **κόστος**. Αφού ελέγχαμε τον παραπάνω ακέραιο σε αυτόν τον πίνακα

#max sum at most λ_μ·A

[0, 0], [1, 0], [2, 0], [3, 0], [4, 0], [5, 0], [6, 0], [7, 0], [8, 0], [9, 0], [10, 10], [11, 10], [12, 10], [13, 10], [14, 10], [15, 15], [16, 15], [17, 17], [18, 18], [19, 18], [20, 18], [21, 18], [22, 18], [23, 18], [24, 18], [25, 25], [26, 26], [27, 27], [28, 28], [29, 28], [30, 30], [31, 30], [32, 32], [33, 33], [34, 33], [35, 35], [36, 36], [37, 37], [38, 37], [39, 37], [40, 40], [41, 41], [42, 42], [43, 43], [44, 44], [45, 45], [46, 45], [47, 47], [48, 48], [49, 48], [50, 50], [51, 51], [52, 52], [53, 53], [54, 54], [55, 55], [56, 56], [57, 57], [58, 58], [59, 59], [60, 60], [61, 61], [62, 62], [63, 63], [64, 64], [65, 65], [66, 66], [67, 67], [68, 68], [69, 69], [70, 70], [71, 71], [72, 72], [73, 73], [74, 74], [75, 75], [76, 76], [77, 77], [78, 78], [79, 79], [80, 80], [81, 81], [82, 82], [83, 83], [84, 84], [85, 85], [86, 86], [87, 87], [88, 88], [89, 89], [90, 90], [91, 91], [92, 92], [93, 93], [94, 94], [95, 95], [96, 96], [97, 97], [98, 98], [99, 99], [100, 100], [101, 101], [102, 102], [103, 103], [104, 104], [105, 105], [106, 106], [107, 107], [108, 108], [109, 109], [110, 110], [111, 111], [112, 112], [113, 113], [114, 114], [115, 115], [116, 116], [117, 117], [118, 118], [119, 119], [120, 120], [121, 121], [122, 122], [123, 123], [124, 124], [125, 125], [126, 126], [127, 127], [128, 128], [129, 129], [130, 130], [131, 131], [132, 132], [133, 133], [134, 134], [135, 135], [136, 136], [137, 137], [138, 138], [139, 139], [140, 140], [141, 141], [142, 142], [143, 143], [144, 144], [145, 145], [146, 145], [147, 147], [148, 148], [149, 148], [150, 150], [151, 151], [152, 152], [153, 153], [154, 154], [155, 155], [156, 155], [157, 155], [158, 158], [159, 159], [160, 160], [161, 160], [162, 162], [163, 163], [164, 163], [165, 165], [166, 165], [167, 167], [168, 168], [169, 169], [170, 170], [171, 170], [172, 170], [173, 170], [174, 170], [175, 170], [176, 170], [177, 177], [178, 178], [179, 178], [180, 180], [181, 180], [182, 180], [183, 180], [184, 180], [185, 185], [186, 185], [187, 185], [188, 185], [189, 185], [190, 185], [191, 185], [192, 185], [193,

Ακολουθούσαμε την παρακάτω διαδικασία:

- Εάν ο ακέραιος ήταν ίδιος μέσα στις αγκύλες τότε πηγαίναμε στην παρακάτω γραμμή του κώδικα με τα διανύσματα



αλλάζοντας μόνο τον αριθμητή του πρώτου διανύσματος και πατώντας enter παίρναμε το cost, το οποίο φαίνεται στην πρώτη τιμή της αγκύλης με τα μπλε γράμματα.

- Εάν ο ακέραιος ήταν διαφορετικός στις παραπάνω αγκύλες τότε θα πηγαίναμε στην ιδιά γραμμή κώδικα αλλάζοντας τους δύο αριθμητές από τα δύο πρώτα διανύσματα, με τις τιμές στις αγκύλες του πίνακα max sum at most λ_μ X A και πατώντας enter βγάζαμε δυο κόστη τα οποία συγκρίναμε και επιλέγαμε το μικρότερο.

2.1.2 Περιγραφή διαδικασίας (MATLAB[1])

Με αυτόν τον τρόπο ολοκληρώνεται η πρώτη πειραματική γραμμή με τα δεδομένα απ' το MAPLE[2] οπότε και οδηγούμαστε στη $2^{η}$ γραμμή με τα αντίστοιχα δεδομένα από το MATLAB[1]. Αφού πρώτα αντιγράφαμε τον κώδικα από το remote pc του καθηγητή και πριν αρχίσει το testing "οδηγούσαμε" το MATLAB[1] στο σωστό path που βρισκόταν ο κώδικας και τρέχοντας την εντολή *load_output* φόρτωνε στην μνήμη του τα απαραίτητα δεδομένα από τον αντίστοιχο κατάλογο *outputHardRough* κάθε πειράματος

Στη συνέχεια το alpha της πρώτης πειραματικής γραμμής το χρησιμοποιούμε σαν input στο MATLAB[1]. Συγκεκριμένα στην συνάρτηση [Vout, tt]= stackoutputprint2(x,x,costall), εάν για παράδειγμα είχαμε σαν alpha από το MAPLE[2] τον αριθμό 0.25567946, κάνουμε αντικατάσταση στη συνάρτηση και την τρέχουμε στη γραμμή εντολών.

```
>> [Vout, tt]= stackoutputprint2(0.25567946 , 0.25567946, costall);
```

Αφού τρέξει για μερικά δευτερόλεπτα, τρέχουμε την εντολή tt στη γραμμή εντολών για να δούμε ότι θα πάρουμε αποτέλεσμα 0.

```
>> [Vout, tt]= stackoutputprint2(0.25567946 , 0.25567946, costall);
>> tt

tt =

     0
```

Σκοπός μας είναι να “βελτιώσουμε” τον αριθμό στο αριστερό μέρος της συνάρτησης όσο το δυνατών περισσότερο ώστε να πάρουμε το μικρότερο - μη μηδενικό – tt. Η διαδικασία φαίνεται στο παρακάτω screenshot, οπού όταν φτάσουμε στον αριθμό 0,255667 η συνάρτηση δεν παίρνει άλλη βελτίωση οπότε σταματάμε.

```
>> [Vout, tt]= stackoutputprint2(0.255661 , 0.25567946, costall);
>> tt

tt =

        2101

>> [Vout, tt]= stackoutputprint2(0.255665 , 0.25567946, costall);
>> tt

tt =

        2101

>> [Vout, tt]= stackoutputprint2(0.255667 , 0.25567946, costall);
>> tt

tt =

        2101

>> [Vout, tt]= stackoutputprint2(0.2556671 , 0.25567946, costall);
>> tt

tt =

     0

>> [Vout, tt]= stackoutputprint2(0.255667 , 0.25567946, costall);
>> tt

tt =

        2101
```

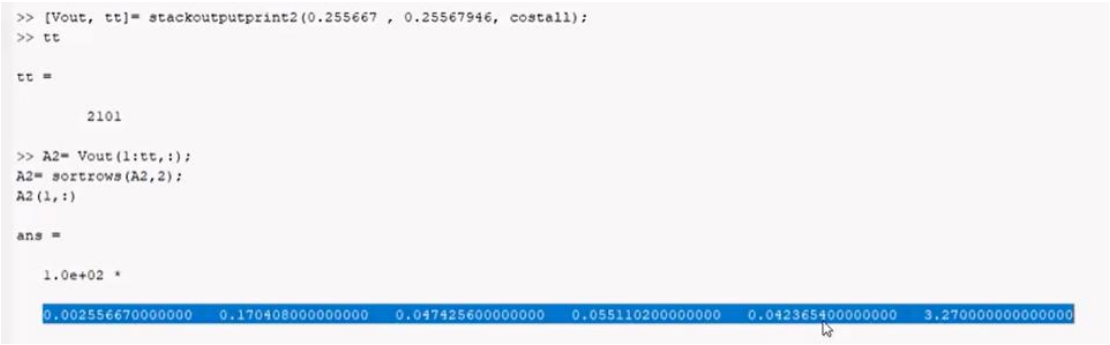
Έπειτα αναζητώντας το ιστορικό εντολών πατώντας το shift επιλέγουμε και τρέχουμε ταυτόχρονα τις 3 παρακάτω εντολές:

A2= Vout(1:tt,:);

A2= sortrows(A2,2);

A2(1,:);

Οπότε το MATLAB[1] μας τυπώνει τη δεύτερη γραμμή του πειράματος μας. Μερικές φορές χρειάζεται η υποδιαστολή να μετακινηθεί 2 θέσεις δεξιά.

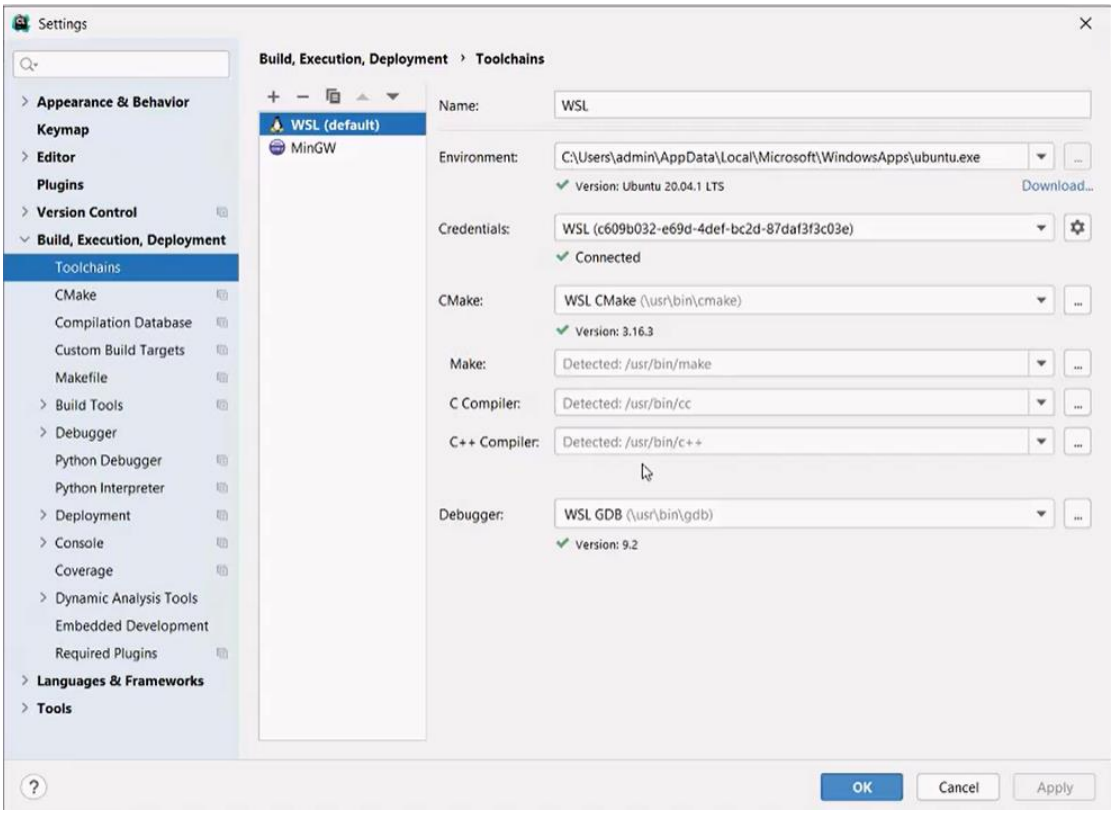


Κατ' αντιστοιχία με το MAPLE[2] έχουμε τον 1ο αριθμό να ναι το alpha τον 2ο να ναι το κόστος, τον 4ο να ναι το dC και τον 5ο να ναι το, ολοκληρώνοντας έτσι και τη 2η πειραματική γραμμή.

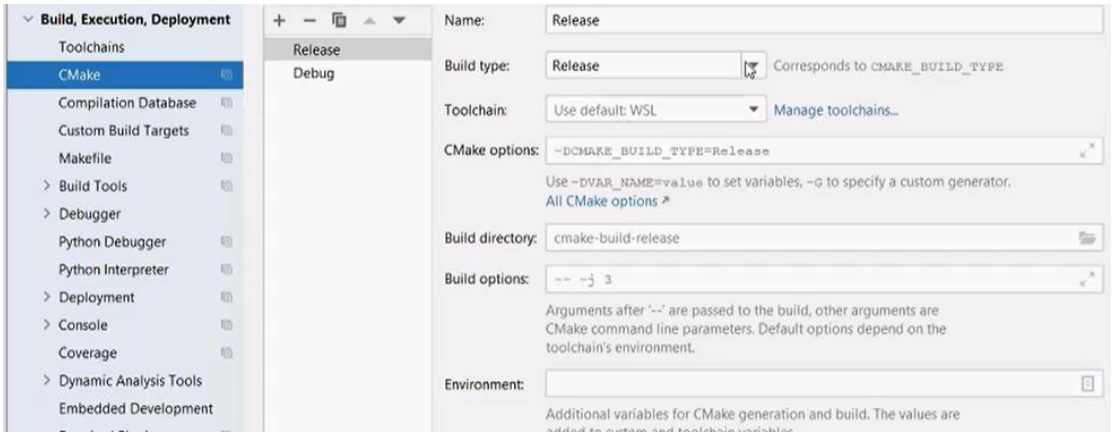
2.1.3 Περιγραφή διαδικασίας (CLion[3])

Για την τρίτη πειραματική γραμμή έπρεπε να εγκαταστήσουμε το Ubuntu[4] σαν virtual machine και το πρόγραμμα CLion[3] στην ελεύθερη φοιτητική έκδοση ώστε να μπορεί το main-parallel.cpp να χρησιμοποιεί βιβλιοθήκες της OpenMP, που είναι απαραίτητες, και να τρέχει παράλληλα threads.

Αρχικά, δημιουργούμε ένα νέο project στη CLion[3] ώστε να φορτώσουμε τον κώδικα και το συγχρονίζουμε με το Ubuntu[4][4] στην παρακάτω καρτέλα των ρυθμίσεων.

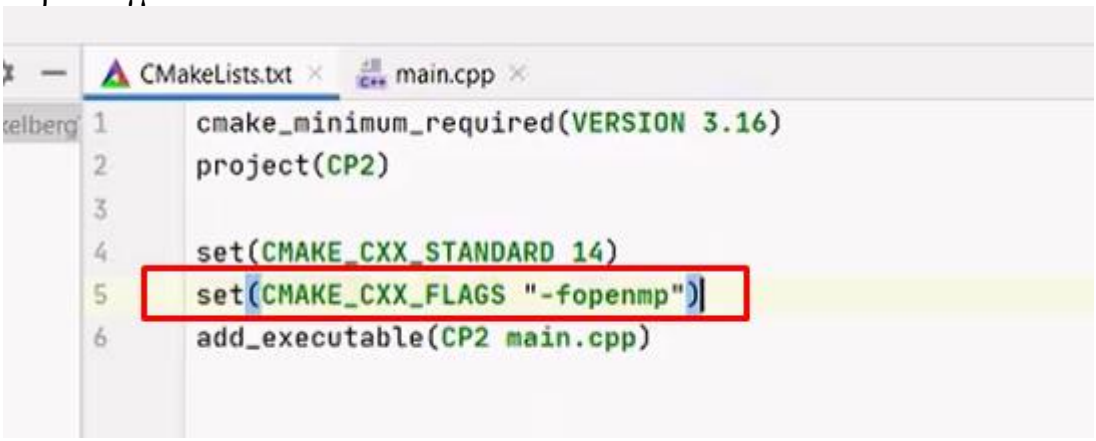


Έπειτα, επιλέγουμε σαν build type το πρόγραμμα μας να κάνει Release.



Σε αυτό το σημείο είμαστε έτοιμοι να φορτώσουμε τον κώδικα και αφού γίνει αυτό πρέπει και να εκμεταλλευτούμε τη βιβλιοθήκη openMP η οποία φαίνεται παρακάτω.

Για να επιτευχθεί αυτό πάμε δίπλα στο CMakeList και γράφουμε την εντολή set(CMAKE_CXX_FLAGS “-fopenmp”) όπως φαίνεται στο παράδειγμα.



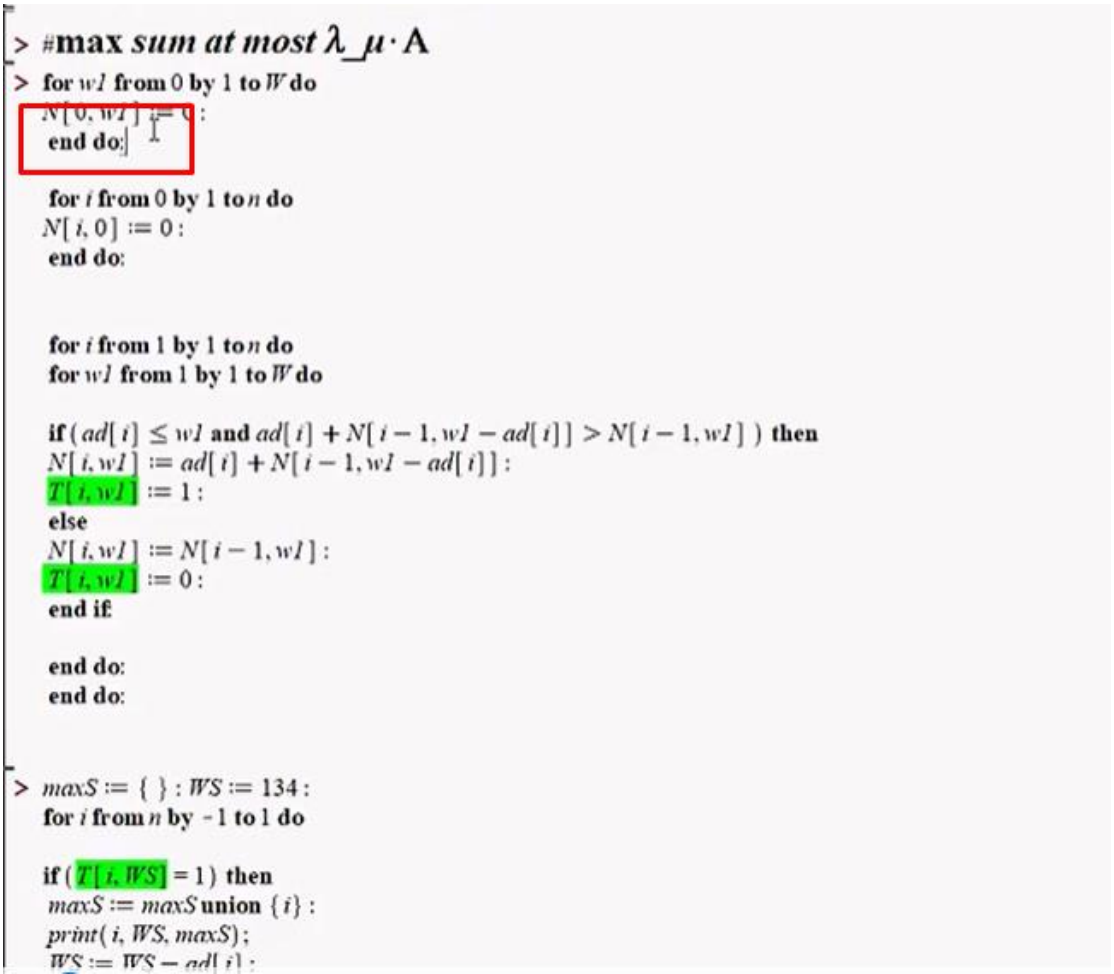
Αμέσως μετά κάνουμε recompile και rebuild all in Release.

Για να τρέξει τώρα αυτό το πρόγραμμα θα πρέπει να διαβάσει ένα input, στην περίπτωση μας το hardKs. Αντιγράφουμε επομένως τον αντίστοιχο φάκελο και τον τοποθετούμε στο release του προγράμματος. Επίσης στον φάκελο αυτό μετά το πέρας ενός πειράματος εμφανίζονται τα 2 αρχεία txt που μας ενδιαφέρουν και τα οποία παίρνουμε σαν αποτελέσματα στην 3η πειραματική γραμμή.

Για να δώσουμε input για το εκάστοτε πείραμα οδηγούμε το Ubuntu[4] με τις κατάλληλες εντολές, στον φάκελο release του προγράμματος μας και αφού μπούμε εκεί, για κάθε πείραμα γράφουμε την εντολή ./CProgram 4.2365400000000 5.5110200000000 0.255667 όπου ./CProgram η ονομασία που δώσαμε στο project, το 4.2365400000000 είναι το L από την 2η γραμμή, το 5.5110200000000 είναι το dC από τη 2η γραμμή και 0.255667 το αντίστοιχο a της 2ης γραμμής του HardRough-Knapsack-vs-NewApproach.txt

2.2 Ολοκλήρωση διαδικασίας

Για την ολοκλήρωση της 3ης γραμμής επιστρέφουμε στο MAPLE[2] στο παρακάτω σημείο του κώδικα και πατάμε enter στο σημείο που φαίνεται



```
> #max sum at most λ_μ·Λ
> for w1 from 0 by 1 to W do
  N[0, w1] := 0 :
  end do;

  for i from 0 by 1 to n do
  N[i, 0] := 0 :
  end do:


  for i from 1 by 1 to n do
  for w1 from 1 by 1 to W do

  if (ad[i] ≤ w1 and ad[i] + N[i − 1, w1 − ad[i]] > N[i − 1, w1]) then
  N[i, w1] := ad[i] + N[i − 1, w1 − ad[i]] :
  T[i, w1] := 1 :
  else
  N[i, w1] := N[i − 1, w1] :
  T[i, w1] := 0 :
  end if:

  end do:
  end do:


> maxS := { } : WS := 134 :
  for i from n by -1 to 1 do

  if (T[i, WS] = 1) then
  maxS := maxS union {i} :
  print(i, WS, maxS);
  WS := WS − ad[i] :
```

Η εντολή αυτή τυπώνει τη μεταβλητή maxS. Μας ενδιαφέρει το σύνολο του φαίνεται παρακάτω.

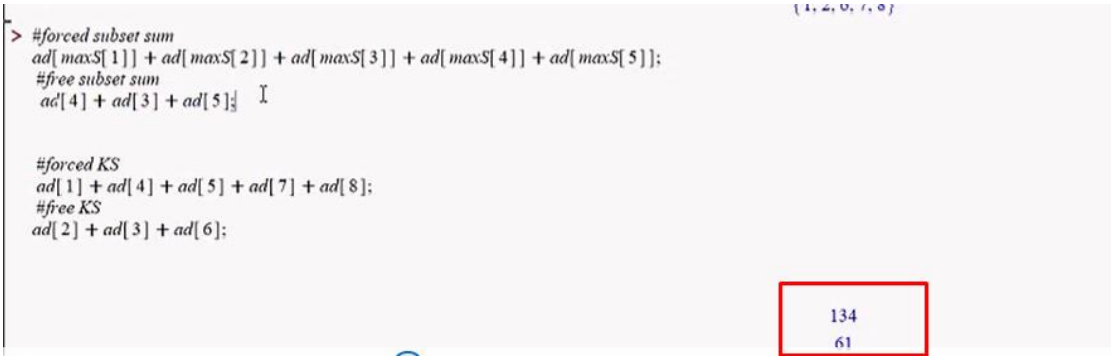
8, 134, {8}
8, 92, {8}
7, 92, {7, 8}
7, 55, {7, 8}
6, 55, {6, 7, 8}
6, 25, {6, 7, 8}
5, 25, {6, 7, 8}
4, 25, {6, 7, 8}
3, 25, {6, 7, 8}
2, 25, {2, 6, 7, 8}
2, 10, {2, 6, 7, 8}
1, 10, {1, 2, 6, 7, 8}
{1, 2, 6, 7, 8}

Στην ενότητα του κώδικα forced subset sum ανάλογα με τα πόσα στοιχεία εμφανίζονται βάζουμε τα αντίστοιχα ad[maxS[..]] στο παράδειγμα μας 5 στοιχεία άρα συμπληρώνουμε μέχρι το ad[maxS[5]],

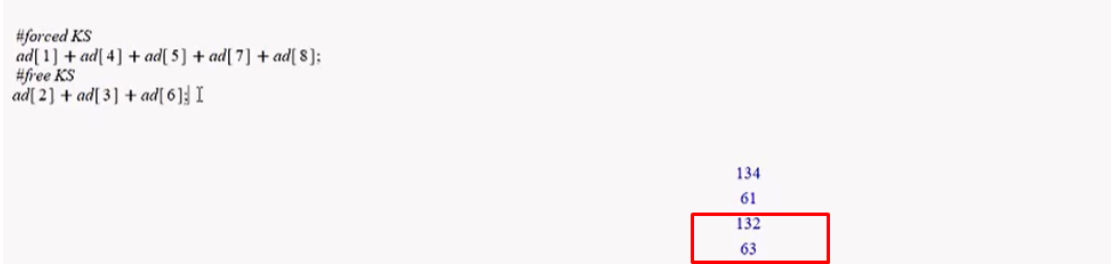
και ομοίως στην από κάτω ενότητα free subset sum βάζουμε τα αντίστοιχα ad[..] αλλά μόνο τα ψηφία που δεν απεικονίζονται στο παραπάνω σύνολο, δηλαδή το παράδειγμα μας τα 3,4,5

```
> #forced subset sum
ad[ maxS[ 1 ] ] + ad[ maxS[ 2 ] ] + ad[ maxS[ 3 ] ] + ad[ maxS[ 4 ] ] + ad[ maxS[ 5 ] ];
 #free subset sum
  ad[ 4 ] + ad[ 3 ] + ad[ 5 ];
```

Πατώντας το enter θα εμφανιστούν τα παρακάτω 134 και 61 που μας ενδιαφέρουν και τα προσθέτουμε στο πείραμα μας



Για να συγκρίνουμε αυτά τα αποτελέσματα με αυτά της CLion[3] παίρνουμε τα αντίστοιχα στοιχεία που προκύπτουν αφού τρέξουμε την εντολή π.χ. ./CProgram 4.2365400000000 5.5110200000000 0.255667 στο Ubuntu[4] και δημιουργούνται 2 αρχεία txt, το forced και το free όπου αντλούμε τα δεδομένα και κατ' αντιστοιχία οδηγούμαστε πάλι στο MAPLE[2] και στην ενότητα forced KS βάζουμε τα δεδομένα από το αντίστοιχο forced ενώ στην ενότητα free KS δεδομένα από το free.txt και πατώντας enter παίρνουμε τα αποτελέσματα όπως φαίνονται παρακάτω.



Αντιγράφοντας τα στο αρχείο HardRough-Knapsack-vs-NewApproach.txt έχουμε συμπληρώσει πλήρως ένα πείραμα

2.3 Συμπεράσματα

Με το πέρας της 1ης πειραματικής αξιολόγησης παιγνιοθεωρητικών στρατηγικών αρχηγού, τα συμπεράσματα που προέκυψαν είναι τα εξής:

- Με τη νέα αυτή προσέγγιση μέσω του MATLAB[1], είδαμε ότι τα κόστη, που είναι και η μεταβλητή που μας ενδιαφέρει περισσότερο, ήταν πολύ κοντά σε σχέση με τα κόστη που MAPLE[2], και πάντα ελαφρός μειωμένα
- Όλες οι υπόλοιπες μεταβλητές (L-dC-Alpha) είναι πολύ κοντά και στις δυο πειραματικές γραμμές από τα δυο προγράμματα, γεγονός που μας επιτρέπει να πούμε πως με παρόμοιες τιμές μπορούμε να έχουμε ακόμα μικρότερο κόστος
- Κατά την εκτέλεση των πειραμάτων στο διάστημα τιμών που προαναφέραμε, σημειωθήκαν οι εξής παρατηρήσεις σχετικά με τον αλγόριθμο στο εκάστοτε πρόγραμμα:
    1. Στο MAPLE[2], αφού είχαμε καταλήξει με τροποποιήσεις και συνέχεις δοκιμές στον κατάλληλο αριθμό, το πρόγραμμα υπολόγιζε και εξήγαγε άμεσα τα αποτελέσματα των μεταβλητών που μας ενδιέφεραν.
    2. Στο MATLAB[1], η συνάρτηση [Vout, tt]= stackoutputprint2(x,x,costall), που χρησιμοποιούσε σαν input τιμές από το MAPLE[2], έβγαζε ακαριαία τα αποτελέσματα της 2ης γραμμής, ωστόσο πολλές φορές χρειάστηκε να πολλαπλασιάσουμε με το 100 όλες τις μεταβλητές καθώς δεν το έκανε σωστά το πρόγραμμα
    3. Τέλος, το ποιο χρονοβόρο πρόγραμμα ήταν αυτό της CLion[3], καθώς όταν εκτελούνταν η εντολή ./CProgram L dC alpha στο Ubuntu[4] υπολόγιζε για πολύ ώρα, μέχρι να εμφανίσει τα δυο txt αρχεία, ενώ παράλληλα χρησιμοποιούσε όλα τα threads του επεξεργαστή και πολύ μεγάλο ποσοστό της μνήμης RAM (~12GB)

Κεφάλαιο 3. Πείραμα 2

3.1.1 Προεργασία

Ξεκινώντας την προεργασία του 2ου πειράματος, από το remote pc, στον οποίο είχαμε πρόσβαση, ο υπεύθυνος καθηγητής κ. Καπόρης δημιούργησε νέο φάκελο που αφορούσε το συγκεκριμένο πείραμα, στο οποίο προστέθηκαν αρχικά καινούρια αρχεία (10) hardKS2 τα οποία αντιγράψαμε και αντικαταστήσαμε στο path που βρισκόντουσαν τα hardKS που αφορούσαν το πρώτο πείραμα και από τα οποία «τραβάει» δεδομένα σαν input μέσω του Fopen η CLion[3].

Για την λειτουργικότητα του MATLAB[1] στο νέο πείραμα, δημιουργούμε νέο φάκελο στον οποίο αντιγράφουμε από το remote PC το νέο outputHardRough4 και οδηγούμε το path του MATLAB[1] στο νέο αυτό φάκελο όπου υπάρχει μέσα το νέο outputHardRough4, επίσης υπάρχει αρχείο κώδικα με όνομα load_output ***προσαρμοσμένο έτσι ώστε να μπορεί το MATLAB[1] να αντλήσει δεδομένα και τέλος ο φάκελος κώδικα με ονομασία stackoutputprint2*** τον οποίο χρησιμοποιούμε σε όλα τα πειράματα. Αφού ολοκληρωθεί αυτή η διαδικασία τρέχουμε την εντολή load_output και περιμένουμε να φορτωθούν τα δεδομένα από τους καταλόγους που υπάρχουν στο φάκελο, όπως ακριβώς και στο πειραμα1. Με το πέρας της διαδικασίας και αφού έχουν γίνει όλα σωστά, το MATLAB[1] είναι έτοιμο να δεχθεί τις κατάλληλες εντολές που θα εξηγηθούν παρακάτω.

Τέλος, με την προεργασία του πειράματος, αντιγράφουμε ένα νέο αρχείο MAPLE[2] με έτοιμο κώδικα κατάλληλο για τον υπολογισμό δεδομένων κάθε πειραματικής γραμμής.

3.1.2 Αρχή πειραματικής διαδικασίας 2ου πειράματος (Εντολές MATLAB[1])

Έχοντας αντικαταστήσει τα καινούρια hardKS ανοίγουμε τον κώδικα της CLion[3] και επιλέγουμε το “rebuild all in Release” έτσι ώστε να προσαρμοστεί το πρόγραμμα στα καινούρια δεδομένα.

Έπειτα είμαστε έτοιμοι να ανοίξουμε το Ubuntu[4] και να το “οδηγήσουμε” στο φάκελο του υπολογιστή μας όπου έχουμε αποθηκευμένα αυτά τα hardKS, στην προκείμενη περίπτωση, στο φάκελο cmake-build-release, όπως φαίνεται παρακάτω.

Σε αυτό ακριβώς το σημείο το πρόγραμμα μας είναι έτοιμο να δεχθεί input, οπότε επιστρέφουμε στο MATLAB[1] για να αντλήσουμε τα δεδομένα που θα χρησιμοποιηθούν γι' αυτό το σκοπό.

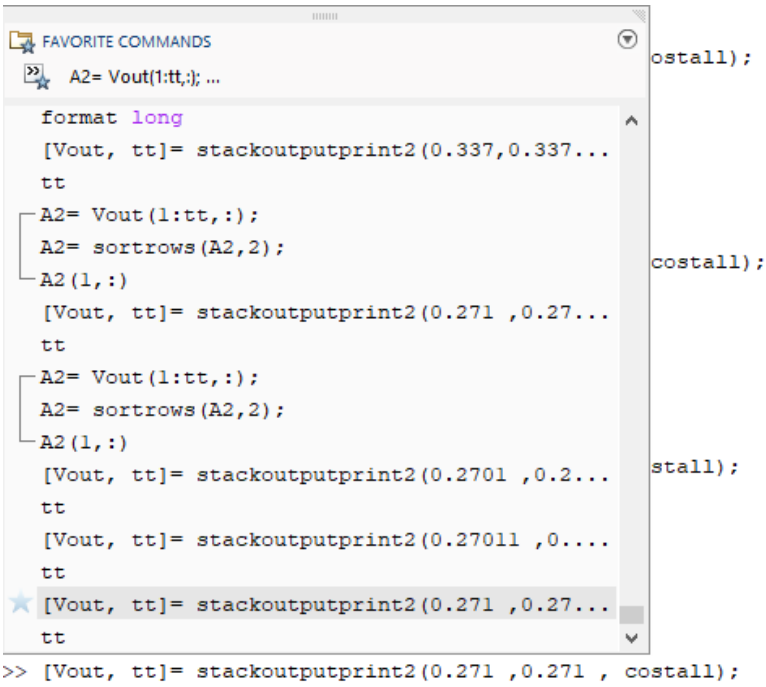
Αφού έχουμε τρέξει την εντολή load_output και το πρόγραμμα μας είναι έτοιμο ξεκινάμε να χρησιμοποιούμε τη συνάρτηση

[Vout, tt]=stackoutputprint2(x,x,costall), όπου x αυτή τη φορά είναι τιμές στο κλειστό διάστημα [0,34 – 0,1]. Μελετάμε όλα τα a μέσα σε αυτό το διάστημα με βήμα 1/100 όπως φαίνεται στο αρχείο που είναι συγκεντρωμένα τα πειραματικά αποτελέσματα.

Θα αναλύσουμε ένα πείραμα εξηγώντας τη γενική μεθοδολογία που ακολουθήθηκε για την ολοκλήρωση κάθε μιας από τις πειραματικές γραμμές

Το παρακάτω παράδειγμα έχει εκτελεσθεί με τυχαίο αριθμό μέσα από το διάστημα, τον αριθμό a = 0.271

Βρισκόμαστε στο περιβάλλον του MATLAB[1] και με το πάνω βελάκι του πληκτρολογίου μας αναζητούμε την εντολή [Vout, tt]= stackoutputprint2(0,271,0,271,costall), εντολή η οποία καλεί την ομώνυμη συνάρτηση stackoutputprint2, και πατάμε το enter.



Αμέσως μετά πληκτρολογούμε την εντολή tt στο command line και παίρνουμε το αποτέλεσμα που αφορά τον αριθμό 0,271

```
>> [Vout, tt]= stackoutputprint2(0.271 ,0.271 , costall);
>> tt

tt =

        2878

>>
```

Τελειώνοντας, επιλέγουμε μαζί τις 3 εντολές

A2= Vout(1:tt,:);

A2= sortrows(A2,2);

A2(1,:);

για να λάβουμε το αποτέλεσμα που μας ενδιαφέρει, όπως φαίνεται στο παρακάτω στιγμιότυπο οθόνης.

```
>> [Vout, tt]= stackoutputprint2(0.271 ,0.271 , costall);
>> tt

tt =

       2878

>> A2= Vout(1:tt,:);
A2= sortrows(A2,2);
A2(1,:)

ans =

   1.0e+02 *

   0.002710000000000   0.174450000000000   0.054243700000000   0.058351800000000   0.042280000000000   3.000000000000000
```

Στην προκείμενη περίπτωση τα αποτελέσματα μας είναι διαιρεμένα με το 100, από bug του προγράμματος οπότε εμείς μετακινούμε την υποδιαστολή 2 θέσεις μέσα.

Από αυτή τη σειρά αποτελεσμάτων που εξάγει το MATLAB[1] θα χρησιμοποιήσουμε σαν input στο Ubuntu[4] για να μας δώσει αποτελέσματα η CLion[3], 2 παραμέτρους, το dC και το L τα οποία παρατίθενται στο παρακάτω screenshot:

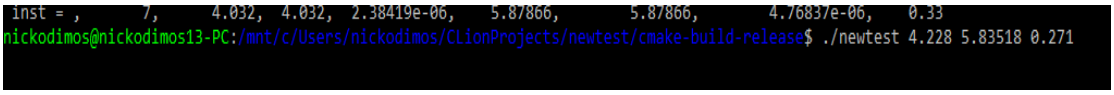
```
ans =

   1.0e+02 *

   0.002710000000000   0.174450000000000   0.054243700000000   0.058351800000000   0.042280000000000   3.000000000000000

>>
```

Βασική προϋπόθεση, να πολλαπλασιάσουμε τους δυο αυτούς αριθμούς με το 100 πριν τους χρησιμοποιήσουμε στο άλλο πρόγραμμα. Τέλος, αποθηκεύουμε τις παραμέτρους dC και L μαζί με το κόστος, το οποίο αναγράφεται στην δεύτερη τιμή του παραπάνω screenshot.

3.1.3 Χρήση δεδομένων από MATLAB[1] σαν input στη CLion[3]

Τελειώνοντας με το MATLAB[1] και έχοντας αντλήσει τα αποτελέσματα που χρειαζόμαστε για τη συνέχεια, μεταφερόμαστε στο Ubuntu[4], το οποίο, όπως αναφέραμε και στην προεργασία του πειράματος έχουμε οδηγήσει στο φάκελο build release του προγράμματος και είναι έτοιμο να δεχθεί το input ώστε να τρέξει και να εξάγει αποτελέσματα η CLion[3].

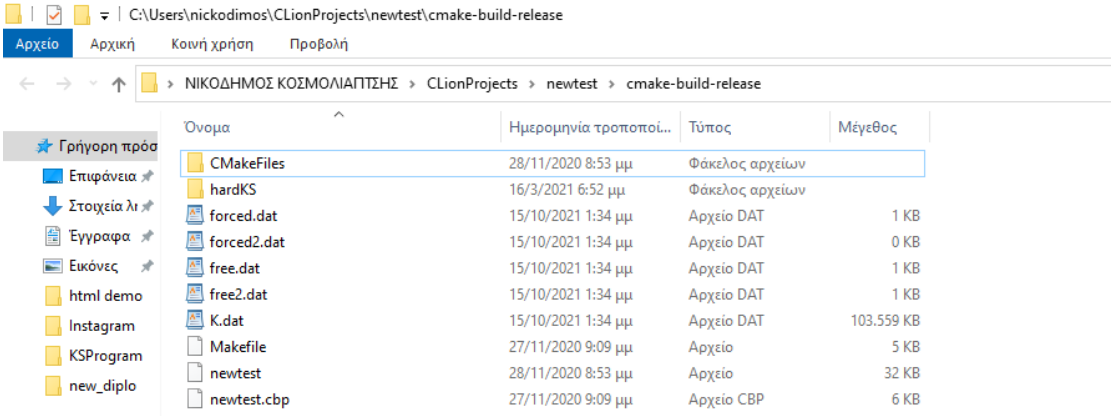
Σε αυτό το σημείο εκτελούμε στο Ubuntu[4] την παρακάτω εντολή:



Αριστερά με τα μπλε γράμματα αναγράφεται το path που οδηγεί στο build release του προγράμματος μας, και παραμένει σταθερό για όλη τη διάρκεια της εκτέλεσης των πειραμάτων. Άμεσος μετά, ακολουθεί η εντολή ./ newtest , όπου newtest το όνομα που δόθηκε στο Project, και έπειτα τρεις αριθμοί. Αυτοί οι αριθμοί με τη σειρά είναι : πρώτα το L από το MATLAB[1] έπειτα το dC από το ίδιο πρόγραμμα και τέλος το εκάστοτε a που τρέχουμε κάθε φορά, στην προκείμενη περίπτωση το 0,271.

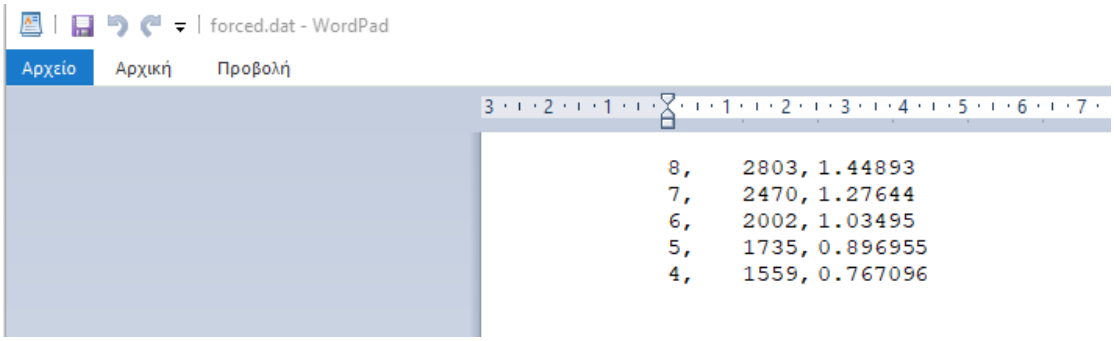
Πατώντας το enter αρχίζει να “τρέχει” ο αλγόριθμος και περιμένουμε λίγα λεπτά ούτος ώστε να πάρουμε τα αποτελέσματα του.

Αφού ολοκληρωθεί η διαδικασία ο κώδικας εξάγει σαν αποτελέσματα πέντε αρχεία στον φάκελο που τον έχουμε οδηγήσει.

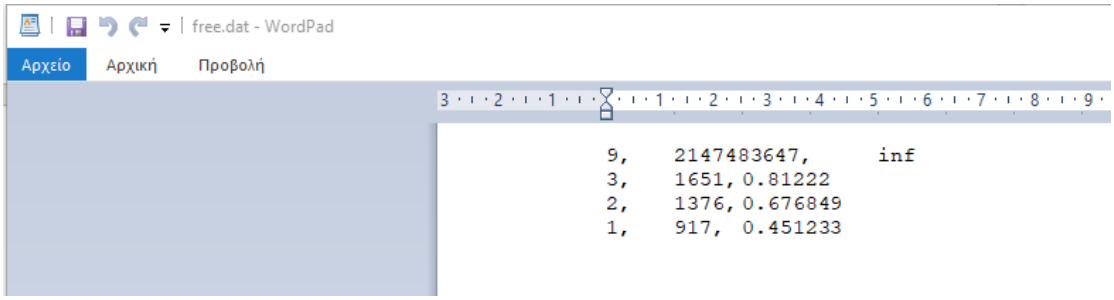


Πρόκειται για τα αρχεία forced και free τα οποία χρησιμοποιούμε αργότερα στον κώδικα του MAPLE[2]

Τα αρχεία αυτά έχουν μέσα αριθμούς από το 1 μέχρι το 9, για παράδειγμα στον αριθμό που εξετάζουμε, το αρχείο forced έβγαλε το συγκεκριμένο αποτέλεσμα



Οπότε αναμένουμε από το αρχείο free να έχει τους αριθμούς από το διάστημα [1,9] που δεν αναγράφονται στο αρχείο forced. Πράγματι όπως φαίνεται παρακάτω αυτό συμβαίνει:



Σε αυτό το στάδιο έχουμε τελειώσει και με το Ubuntu[4] – CLion[3] και έχουμε ότι χρειαζόμαστε για να μεταφερθούμε στο 3ο και τελευταίο στάδιο, στο αρχείο κώδικα του MAPLE[2].

3.1.4 Ολοκλήρωση πειραματικής γραμμής μέσω MAPLE[2]

Ο κώδικας αυτός σε μορφή ψευδοκώδικα φαίνεται παρακάτω:

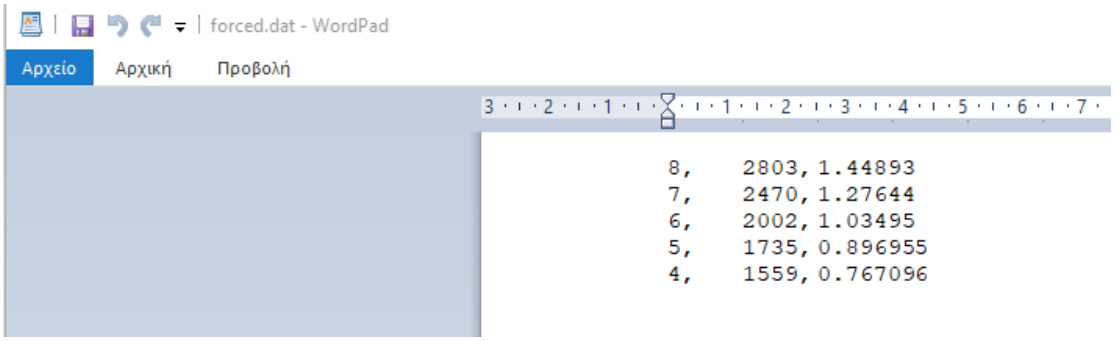
```
d = 100

n = 8

a = [0.1, 0.15, 0.18, 0.17, 0.26, 0.3, 0.37, 0.42]

ad = a * d

A = sum of a from i = 1 to 8

Ad = sum of ad from i = 1 to 8

Ad = round down Ad to nearest integer

W = Ad

for i = 1 to n

   ad[i] = round down ad[i] to nearest integer

end for
```

```
p1 = 1 ,p2 = 0,p3 = 1,p4 = 0,p5 = 1,

p6 = 0,p7 = 1,p8 = 0

#Forced KS

result1 = p1 * ad[1] + p2 * ad[2] + p3 * ad[3] +
p4 * ad[4]

 + p5 * ad[5] + p6 * ad[6] + p7 * ad[7] + p8 *
ad[8]

#Free KS

result2 = (1 - p1) * ad[1] + (1 - p2) * ad[2] + (1 -
p3) * ad[3]
 + (1 - p4) * ad[4] + (1 - p5) * ad[5] + (1 - p6) *
ad[6] +
(1 - p7) * ad[7] + (1 - p8) * ad[8]
```

Το κομμάτι του κώδικα MAPLE[2] που επεξεργαζόμαστε σε κάθε πείραμα είναι οι τιμές που βρίσκονται κάτω από το σχόλιο #forcedKS.

Εκεί υπάρχουν οκτώ pi, οπού i οι ακέραιοι αριθμοί που ανήκουν στο διάστημα [1,8], και δέχονται μόνο τιμές 0 και 1.

Για να καθοριστεί το τι τιμές θα πάρουν αυτά τα p1,p2,…,p8, δηλαδή αν θα έχουν 0 η 1, αρκεί να εξετάσουμε το αρχείο forced που εξάγει η CLion[3] και να βάλουμε άσσους στα αντίστοιχα pi των αριθμών που αναγράφονται στο αρχείο



Δηλαδή στην προκείμενη περίπτωση βάζουμε άσσους στα εξής:

p4 = 1

p5 = 1

p6 = 1

p7 = 1

p8 = 1

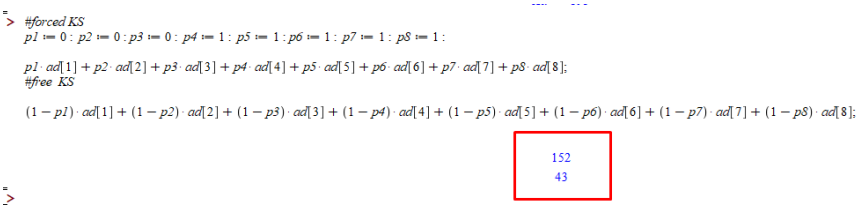
και στα υπόλοιπα τοποθετούμε το 0 δηλαδή

p1 = 0

p2 = 0

p3 = 0

Αφού συμπληρώσουμε τις τιμές καθ' αυτόν τον τρόπο είμαστε έτοιμοι να πατήσουμε το enter και ο αλγόριθμος να δουλέψει ελάχιστα δευτερόλεπτα και να πάρουμε το αποτέλεσμα που χρειαζόμαστε το οποίο είναι το 152,43.



Οι δυο αυτοί αριθμοί ,152 και 43 στην προκείμενη περίπτωση, και γενικά σε όλα τα πειράματα πρέπει να πούμε πως θα έχουν πάντα άθροισμα το 195 και είναι ένας τρόπος να επαληθεύσουμε ότι έχουμε κάνει σωστά τη διαδικασία.

Αποθηκεύοντας και αυτά τα τελευταία δεδομένα στο συγκεντρωτικό αρχείο μας, έχει ολοκληρωθεί πλήρως μια πειραματική γραμμή και καθ' αυτόν τον τρόπο συνεχίζουμε μέχρις ότου να ολοκληρώσουμε τους αριθμούς μέσα στο διάστημα αυτό.

Το συγκεντρωτικό μας αρχείο είναι αυτής της μορφής:

```
b= 4+0.5
-------------------------------------------------------------------------
a          {forced}{free}[sum - sum]    cost              dC       L
-------------------------------------------------------------------------

0.278 -    {1,2,5,6,7,8}{3,4,9}[160-30] 17.3890           5.77754  4.216
0.277 -    {1,4,5,6,7,8}{2,3,9}[162-33] 17.3894           5.75067  4.224
0.276 -    {1,4,5,6,7,8}{2,3,9}[162-33] 17.3952           5.74595  4.228
0.275 -    {1,3,5,6,7,8}{2,4,9}[163-32] 17.4040           5.73036  4.236
0.274 -    {1,3,5,6,7,8}{2,4,9}[163-32] 17.4098           5.72564  4.240
0.273 -    {1,3,5,6,7,8}{2,4,9}[163-32] 17.4157           5.72092  4.244
0.272 -    {2,3,5,6,7,8}{1,4,9}[168-27] 17.4200           5.66513  4.260
0.271 -    {4,5,6,7,8}{1,2,3,9}[152-43] 17.4450           5.83518  4.228
0.270 -    {1,4,5,6,7,8}{2,3,9}[162-33] 17.4396           5.71702  4.256
0.269 -    {1,4,5,6,7,8}{2,3,9}[162-33] 17.4458           5.71231  4.260
0.268 -    {4,5,6,7,8}{1,2,3,9}[152-43] 17.4482           5.81959  4.236
0.267 -    {1,4,5,6,7,8}{2,3,9}[162-33] 17.4449           5.70246  4.264
0.266 -    {1,4,5,6,7,8}{2,3,9}[162-33] 17.4512           5.69774  4.268
0.265 -    {1,3,5,6,7,8}{2,4,9}[163-32] 17.4730           5.68256  4.280
0.264 -    {2,4,5,6,7,8}{1,3,9}[167-28] 17.4748           5.63764  4.292
0.263 -    {2,5,6,7,8}{1,3,4,9}[150-45] 17.5033           5.81795  4.260
0.262 -    {2,4,5,6,7,8}{1,3,9}[167-28] 17.4869           5.62820  4.300
0.261 -    {3,5,6,7,8}{1,2,4,9}[150-45] 17.4792           5.77220  4.264
0.26  -    {1,4,5,6,7,8}{2,3,9}[163-32] 17.4988           5.66882  4.296
0.259 -    {1,2,5,6,7,8}{3,4,9}[160-30] 17.4959           5.68482  4.292
```

3.2. Συμπεράσματα

Το παρών 2ο πείραμα αξιολόγησης παιγνιοθεωρητικών στρατηγικών αρχηγού, ήταν αρκετά διαφορετικό από το πρώτο για τους λογούς που θα αναλύσουμε παρακάτω:

- Το σύνολο των αποτελεσμάτων αποτελούν μια πειραματική γραμμή, γεγονός που μας οδηγεί στο να μην συγκρίνουμε τιμές μεταξύ τους, αλλά στο να έχουμε μια εικόνα για την συμπεριφορά των δεδομένων στο εύρος τιμών που μελετάμε.
- Για την ολοκλήρωση μιας πειραματικής γραμμής, συλλέγουμε δεδομένα και από τα τρία προγράμματα (cost – dC – L από MATLAB[1], και τα forced – sum από τα MAPLE[2] & CLion[3]) ενώ στο 1ο πείραμα κάθε γραμμή από δεδομένα αποτελούνταν και από διαφορετικό πρόγραμμα.
- Και σε αυτήν την περίπτωση σημειωθήκαν παρατηρήσεις σχετικά με τον αλγόριθμο στο εκάστοτε πρόγραμμα:
    1. Στο MATLAB[1], η συνάρτηση [Vout, tt]= stackoutputprint2(x,x,costall), που χρησιμοποιούσε σαν input τιμές από το κλειστό διάστημα πραγματικών τιμών που μελετούσαμε, έβγαζε ακαριαία τα αποτελέσματα της, ωστόσο η εντολή **tt**, καθυστερούσε ελαφρώς. Και εδώ, πολλές φορές χρειάστηκε να πολλαπλασιάσουμε με το 100 όλες τις μεταβλητές για να πάρουμε το σωστό αποτέλεσμα, καθώς δεν το έκανε σωστά το πρόγραμμα
    2. Και στο 2ο πείραμα, το ποιο χρονοβόρο πρόγραμμα ήταν αυτό της CLion[3], καθώς όταν εκτελούνταν η εντολή ./newtest L dC alpha στο Ubuntu[4] “έτρεχε” για αρκετή ώρα, μέχρι να εμφανίσει τα δυο txt αρχεία, ενώ παράλληλα χρησιμοποιούσε όλα τα threads του επεξεργαστή και πολύ μεγάλο ποσοστό της μνήμης RAM (~10GB)
    3. Τέλος, στο MAPLE[2], αφού τοποθετούσαμε τους άσους και τα μηδενικά με τον τρόπο που προαναφέραμε στην συνάρτηση, λαμβάναμε το αποτέλεσμα σχεδόν άμεσα χωρίς καθυστερήσεις.

Κεφάλαιο 4. Πείραμα 3

4.1.1 Προεργασία

Σχετικά με την προετοιμασία του τρίτου πειράματος, μεταφέραμε από μέσω του remote pc,όπου είχαμε πρόσβαση στον Η/Υ του υπεύθυνου καθηγητή κ. Καπόρη, τα καινούρια αρχεία **HardKS** με τους υπό φακέλους hard1,hard2,..,hard10 από όπου τραβάει μέσω του Fopen δεδομένα το πρόγραμμα στην CLion[3],το **outputHardRough6** από το οποίο τραβάει δεδομένα το MATLAB[1] με το νέο load_output.m, και το **Experiment-6_154_1000_10000-results.txt** στο οποίο θα συμπληρώνουμε γραμμή γραμμή τα δεδομένα από τις συγκρίσεις.

Για την λειτουργικότητα του MATLAB[1] στο νέο πείραμα, δημιουργούμε νέο φάκελο στον οποίο αντιγράφουμε από το remote PC το νέο outputHardRough6 και οδηγούμε το path του MATLAB[1] στο νέο αυτό φάκελο όπου υπάρχει μέσα το νέο **outputHardRough6**.Επίσης υπάρχει αρχείο κώδικα με όνομα **load_output.m** προσαρμοσμένο έτσι ώστε να μπορεί το MATLAB[1] να αντλήσει δεδομένα από το **outputHardRough6** και τέλος το αρχείο κώδικα με ονομασία **stackoutputprint2** τον οποίο χρησιμοποιούμε σε όλα τα πειράματα. Αφού ολοκληρωθεί αυτή η διαδικασία τρέχουμε την εντολή **load_output** και περιμένουμε να φορτωθούν τα δεδομένα από τους καταλόγους που υπάρχουν στο φάκελο, όπως ακριβώς και στο πείραμα 2. Με το πέρας της διαδικασίας και αφού έχουν γίνει όλα σωστά, το MATLAB[1] είναι έτοιμο να δεχθεί τις κατάλληλες εντολές που θα εξηγηθούν παρακάτω.

Τέλος, με την προεργασία του πειράματος, αντιγράφουμε το αρχείο κώδικα **forced-sum** της MAPLE[2] από το προηγούμενο πείραμα κατάλληλο για τον υπολογισμό δεδομένων κάθε πειραματικής γραμμής.

4.1.2 Αρχή πειραματικής διαδικασίας 3ου πειράματος. (MATLAB[1]&Ubuntu[4])

Έχοντας αντικαταστήσει τα καινούρια **hardKS** ανοίγουμε τον κώδικα της CLion[3] και επιλέγουμε το "**rebuild all in Release**" έτσι ώστε να προσαρμοστεί το πρόγραμμα στα καινούρια δεδομένα.

Έπειτα είμαστε έτοιμοι να ανοίξουμε το Ubuntu[4]. Κάνουμε login με την εντολή **ssh icsd13190@localhost -p2222** (το όνομα χρήστη αλλάζει ανάλογα ποιος τρέχει το πείραμα) έτσι ώστε να συνδεθούμε στο **wls Toolchain**.Έπειτα μέσω της εντολής **cd** το οδηγούμε στο φάκελο του υπολογιστή μας όπου έχουμε αποθηκευμένα αυτά τα **hardKS**, στην προκείμενη περίπτωση, στο φάκελο **cmake-build-release**, όπως φαίνεται παρακάτω.

```
icsd13190@DESKTOP-39OB77P:~$ cd /mnt/f/diplwmatikh/Experiment-6/prg/cmake-build-release
```

Το συγκεκριμένο πείραμα θα αποτελείται από 3 γραμμές αποτελεσμάτων για κάθε **alpha**. Για την δημιουργία της πρώτης γραμμής αποτελεσμάτων η οποία βρίσκεται στο κέντρο ακολουθούμε την εξής διαδικασία.

Αφού έχουμε τρέξει την εντολή **load_output** στο MATLAB[1] και τα δεδομένα μας έχουν φορτωθεί, το πρόγραμμα μας είναι έτοιμο και ξεκινάμε χρησιμοποιώντας τη συνάρτηση **[Vout, tt]=stackoutputprint2(x,x,costall)**, όπου x είναι τιμές στο κλειστό διάστημα [0,336 – 0,05]. Μελετάμε όλα τα a μέσα σε αυτό το διάστημα με βήμα 1/100 όπως φαίνεται στο αρχείο που είναι συγκεντρωμένα τα πειραματικά αποτελέσματα.

Θα αναλύσουμε ένα πείραμα εξηγώντας τη γενική μεθοδολογία που ακολουθήθηκε για την ολοκλήρωση κάθε μιας από τις πειραματικές γραμμές

Το παρακάτω παράδειγμα έχει εκτελεσθεί με τυχαίο αριθμό μέσα από το διάστημα, τον αριθμό a = 0.260.

Βρισκόμαστε στο περιβάλλον του MATLAB[1] και αρχικά πληκτρολογούμε την εντολή **format long** για το format των αριθμών. Έπειτα με το πάνω βελάκι του πληκτρολογίου μας αναζητούμε την εντολή **[Vout,tt]=stackoutputprint2(x,x,costall)**, εντολή η οποία καλεί την ομώνυμη συνάρτηση **stackoutputprint2**,όπου ως x βάζουμε το a = 0.260 και πατάμε το enter. Αμέσως μετά πληκτρολογούμε την εντολή **tt**

```
>> load_output
>> format long
>> [Vout, tt]= stackoutputprint2(0.260 , 0.260 , costall);
>> tt

tt =

        6125

>>
```

στο command line και παίρνουμε το αποτέλεσμα που αφορά τον αριθμό 0,260.

Τελειώνοντας, επιλέγουμε μαζί τις 3 εντολές

A2= Vout(1:tt,:);

A2= sortrows(A2,2);

A2(1,:);

για να λάβουμε το αποτέλεσμα που μας ενδιαφέρει, όπως φαίνεται στο παρακάτω στιγμιότυπο οθόνης.

```
>> A2= Vout(1:tt,:);
A2= sortrows(A2,2);
A2(1,:)

ans =

   1.0e+02 *

                        Cost                                dC                  L

   0.002600000000000   0.173248000000000   0.050421500000000   0.056780800000000   0.042559500000000   2.750000000000000

fx >>
```

Στην προκείμενη περίπτωση τα αποτελέσματα μας είναι διαιρεμένα με το 100, από bug του προγράμματος οπότε εμείς μετακινούμε ώστε να μεταφερθεί η υποδιαστολή 2 θέσεις μέσα.

Από αυτή τη σειρά αποτελεσμάτων που εξάγει το MATLAB[1] θα χρησιμοποιήσουμε σαν input στο Ubuntu[4] για να μας δώσει αποτελέσματα η CLion[3], 2 παραμέτρους, το **dC** και το **L**.

Βασική προϋπόθεση, να πολλαπλασιάσουμε τους δυο αυτούς αριθμούς με το 100 πριν τους χρησιμοποιήσουμε στο άλλο πρόγραμμα. Τέλος, αποθηκεύουμε τις παραμέτρους **dC** και **L** μαζί με το **cost**, το οποίο αναγράφεται στην δεύτερη τιμή του παραπάνω screenshot.

Τελειώνοντας με το MATLAB[1] και έχοντας αντλήσει τα αποτελέσματα που χρειαζόμαστε για τη συνέχεια, μεταφερόμαστε στο Ubuntu[4], το οποίο, όπως αναφέραμε και προηγουμένως, έχουμε οδηγήσει στο φάκελο **cmake-build-release** του προγράμματος και είναι έτοιμο να δεχθεί το input ώστε να τρέξει και να εξάγει αποτελέσματα η CLion[3].

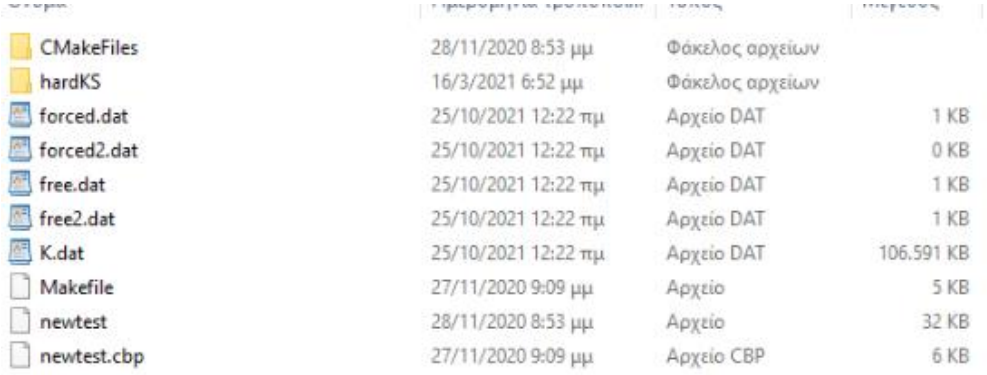
Σε αυτό το σημείο εκτελούμε στο Ubuntu[4] την παρακάτω εντολή:

```
icsd13190@DESKTOP-39OB77P:/mnt/f/diplwmatikh/Experiment-6/prg/cmake-build-release$ ./prg 4.25595 5.67808 0.260
```

Αριστερά με τα μπλε γράμματα αναγράφεται το path που οδηγεί στο build release του προγράμματος μας, και παραμένει σταθερό για όλη τη διάρκεια της εκτέλεσης των πειραμάτων. Έπειτα, ακολουθεί η εντολή **./prg** , όπου prg το όνομα που δόθηκε στο Project εμπνευσμένο από την αγγλική λέξη program, και έπειτα τρεις αριθμοί. Αυτοί οι αριθμοί με τη σειρά είναι : πρώτα το **L** από το MATLAB[1] έπειτα το **dC** από το ίδιο πρόγραμμα και τέλος το εκάστοτε **alpha** που τρέχουμε κάθε φορά, στην προκείμενη περίπτωση το 0,260.

Πατώντας το enter αρχίζει να "τρέχει" ο αλγόριθμος και περιμένουμε λίγα λεπτά ούτος ώστε να πάρουμε τα αποτελέσματα του.

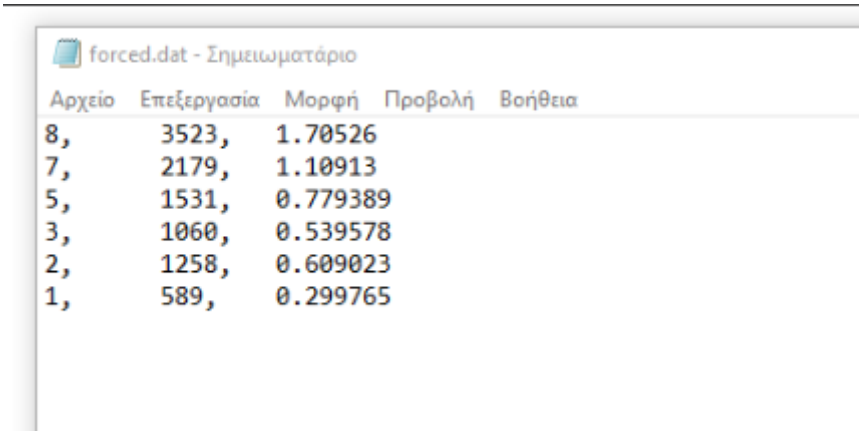
Αφού ολοκληρωθεί η διαδικασία ο κώδικας εξάγει σαν αποτελέσματα πέντε αρχεία στον φάκελο που τον έχουμε οδηγήσει.



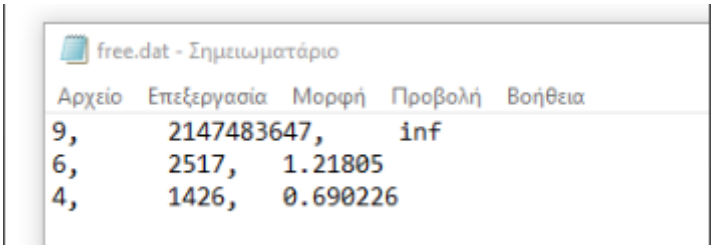
CMakeFiles	28/11/2020 8:53 μμ	Φάκελος αρχείων	
hardKS	16/3/2021 6:52 μμ	Φάκελος αρχείων	
forced.dat	25/10/2021 12:22 πμ	Αρχείο DAT	1 KB
forced2.dat	25/10/2021 12:22 πμ	Αρχείο DAT	0 KB
free.dat	25/10/2021 12:22 πμ	Αρχείο DAT	1 KB
free2.dat	25/10/2021 12:22 πμ	Αρχείο DAT	1 KB
K.dat	25/10/2021 12:22 πμ	Αρχείο DAT	106.591 KB
Makefile	27/11/2020 9:09 μμ	Αρχείο	5 KB
newtest	28/11/2020 8:53 μμ	Αρχείο	32 KB
newtest.cbp	27/11/2020 9:09 μμ	Αρχείο CBP	6 KB

Πρόκειται για τα αρχεία **forced** και **free** τα οποία χρησιμοποιούμε αργότερα στον κώδικα του MAPLE[2],όπως και για το αρχείο K

Τα αρχεία **forced** και **free** εμπεριέχουν αριθμούς από το 1 μέχρι το 9, για παράδειγμα στον αριθμό που εξετάζουμε, το αρχείο **forced** έβγαλε το συγκεκριμένο αποτέλεσμα,



Οπότε αναμένουμε από το αρχείο **free** να έχει τους αριθμούς από το διάστημα [1,9] που δεν αναγράφονται στο αρχείο **forced**. Πράγματι όπως φαίνεται παρακάτω αυτό συμβαίνει:



4.1.3 Χρήση του output της CLion[3] σαν input στο MAPLE[2]

Σε αυτό το στάδιο έχουμε τελειώσει και με το Ubuntu[4] – CLion[3] και έχουμε ότι χρειαζόμαστε για να μεταφερθούμε στο 3ο και τελευταίο στάδιο της πρώτης γραμμής των πειραμάτων, στο αρχείο κώδικα του MAPLE[2].

Ο κώδικας αυτός σε μορφή ψευδοκώδικα φαίνεται παρακάτω:

```
d = 100

n = 8

a = [0.1, 0.15, 0.18, 0.17, 0.26, 0.3, 0.37, 0.42]

ad = a * d

A = sum of a from i = 1 to 8

Ad = sum of ad from i = 1 to 8

Ad = round down Ad to nearest integer

W = Ad

for i = 1 to n

   ad[i] = round down ad[i] to nearest integer

end for
```

```
p1 = 1 ,p2 = 0,p3 = 1,p4 = 0,p5 = 1,

p6 = 0,p7 = 1,p8 = 0

#Forced KS

result1 = p1 * ad[1] + p2 * ad[2] + p3 * ad[3] +
p4 * ad[4]

 + p5 * ad[5] + p6 * ad[6] + p7 * ad[7] + p8 *
ad[8]

#Free KS

result2 = (1 - p1) * ad[1] + (1 - p2) * ad[2] + (1 -
p3) * ad[3]
 + (1 - p4) * ad[4] + (1 - p5) * ad[5] + (1 - p6) *
ad[6] +
(1 - p7) * ad[7] + (1 - p8) * ad[8]
```

Το κομμάτι του κώδικα MAPLE[2] που επεξεργαζόμαστε σε κάθε πείραμα είναι οι τιμές που βρίσκονται κάτω από το σχόλιο #**forcedKS**.

Εκεί υπάρχουν οκτώ pi, οπού i οι ακέραιοι αριθμοί που ανήκουν στο διάστημα [1,8], και δέχονται μόνο τιμές 0 και 1.

Για να καθοριστεί το τι τιμές θα πάρουν αυτά τα p1,p2,…,p8, δηλαδή αν θα έχουν 0 η 1, αρκεί να εξετάσουμε το αρχείο forced που εξάγει η CLion[3] και να βάλουμε άσσους στα αντίστοιχα pi των αριθμών που αναγράφονται στο αρχείο.

Δηλαδή στην προκείμενη περίπτωση βάζουμε άσσους στα εξής:

p1 = 1

p2 = 1

p3 = 1

p5 = 1

p7 = 1

p8 = 1

και στα υπόλοιπα τοποθετούμε το 0 δηλαδή

p4 = 0

p6 = 0

Αφού συμπληρώσουμε τις τιμές καθ’ αυτόν τον τρόπο είμαστε έτοιμοι να πατήσουμε το enter και ο αλγόριθμος να δουλέψει ελάχιστα

δευτερόλεπτα και να πάρουμε το αποτέλεσμα που χρειαζόμαστε το οποίο είναι το 148,47.

```
#forced KS
p1 := 1 : p2 := 1 : p3 := 1 : p4 := 0 : p5 := 1 : p6 := 0 : p7 := 1 : p8 := 1 :
p1 · ad[1] + p2 · ad[2] + p3 · ad[3] + p4 · ad[4] + p5 · ad[5] + p6 · ad[6] + p7 · ad[7] + p8 · ad[8];
#free KS
(1 − p1) · ad[1] + (1 − p2) · ad[2] + (1 − p3) · ad[3] + (1 − p4) · ad[4] + (1 − p5) · ad[5] + (1 − p6) · ad[6] + (1 − p7) · ad[7] + (1 − p8) · ad[8];
                                                                                                                    148
                                                                                                                    47
```

Οι δυο αυτοί αριθμοί ,148 και 47 στην προκείμενη περίπτωση, και γενικά σε όλα τα πειράματα πρέπει να πούμε πως θα έχουν πάντα άθροισμα το 195 και είναι ένας τρόπος να επαληθεύσουμε ότι έχουμε κάνει σωστά τη διαδικασία.

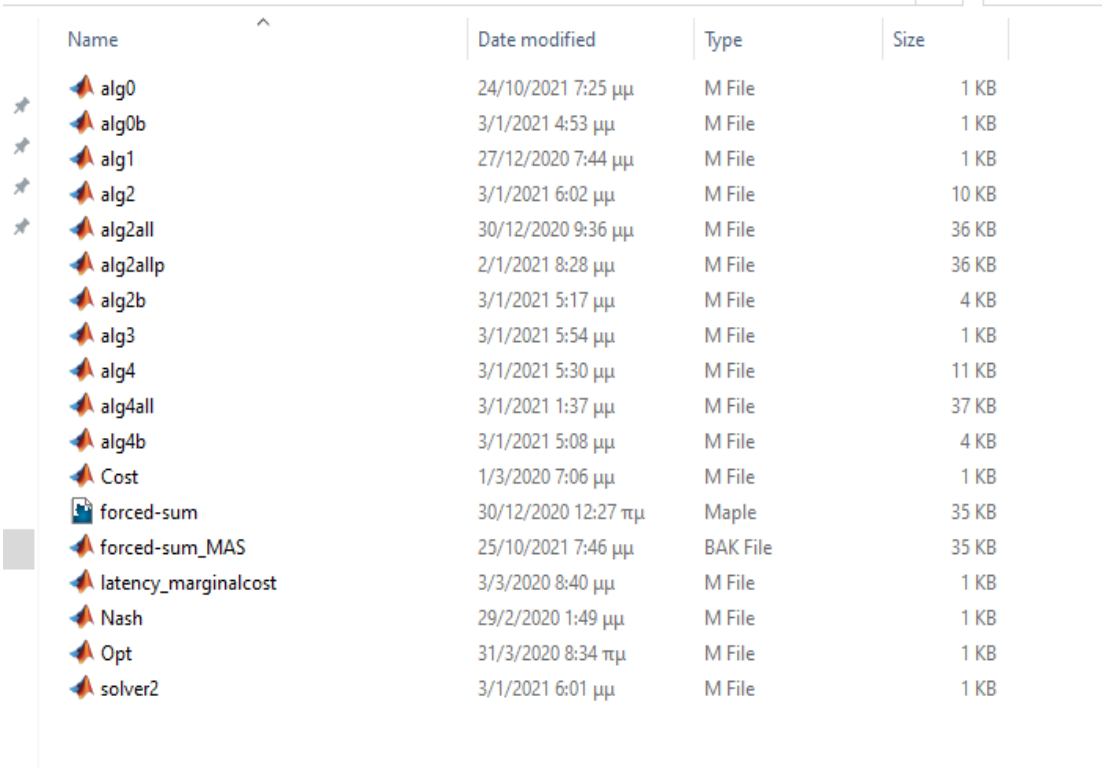
Αποθηκεύοντας και αυτά τα τελευταία δεδομένα στο συγκεντρωτικό αρχείο μας, έχει ολοκληρωθεί πλήρως την πρώτη πειραματική γραμμή και καθ' αυτόν τον τρόπο συνεχίζουμε μέχρις ότου να ολοκληρώσουμε τους αριθμούς μέσα στο διάστημα αυτό.

```
0.260                {1,2,3,5,7,8}{4,6,9}[148-47]          17.3248          5.67808          4.25595
```

4.2.1 2η & 3η πειραματική γραμμή (MATLAB[1])

Αφού ολοκληρώσαμε την 1η γραμμή δεδομένων του πειράματος, συνεχίζουμε ώστε να συμπληρώσουμε και τις υπολειπόμενες δυο γραμμές.

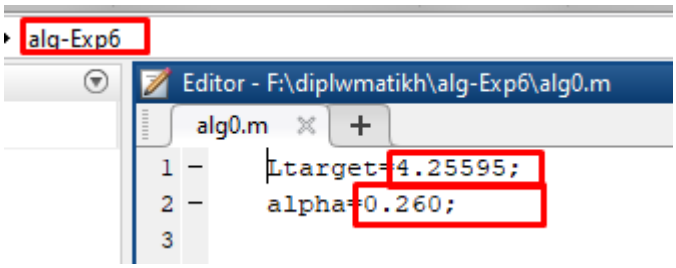
Στην προκειμένη περίπτωση θα χρησιμοποιήσουμε μόνο τα προγράμματα MATLAB[1] και MAPLE[2].Δημιουργούμε ένα νέο



Name	Date modified	Type	Size
alg0	24/10/2021 7:25 μμ	M File	1 KB
alg0b	3/1/2021 4:53 μμ	M File	1 KB
alg1	27/12/2020 7:44 μμ	M File	1 KB
alg2	3/1/2021 6:02 μμ	M File	10 KB
alg2all	30/12/2020 9:36 μμ	M File	36 KB
alg2allp	2/1/2021 8:28 μμ	M File	36 KB
alg2b	3/1/2021 5:17 μμ	M File	4 KB
alg3	3/1/2021 5:54 μμ	M File	1 KB
alg4	3/1/2021 5:30 μμ	M File	11 KB
alg4all	3/1/2021 1:37 μμ	M File	37 KB
alg4b	3/1/2021 5:08 μμ	M File	4 KB
Cost	1/3/2020 7:06 μμ	M File	1 KB
forced-sum	30/12/2020 12:27 πμ	Maple	35 KB
forced-sum_MAS	25/10/2021 7:46 μμ	BAK File	35 KB
latency_marginalcost	3/3/2020 8:40 μμ	M File	1 KB
Nash	29/2/2020 1:49 μμ	M File	1 KB
Opt	31/3/2020 8:34 πμ	M File	1 KB
solver2	3/1/2021 6:01 μμ	M File	1 KB

φάκελο με το όνομα **alg-Exp6** στον οποίο μεταφέρουμε από το υπολογιστή του κ Καπόρη μέσω του remote PC ένα σύνολο αρχείων προγραμμάτων MATLAB[1] όπως και το **forced-sum** που χρησιμοποιήσαμε προηγουμένως.

Αρχικά ανοίγουμε το αρχείο **alg0.m** μέσω του MATLAB[1] στο οποίο θα εισάγουμε τα δεδομένα με την επεξεργασία των δύο πρώτων μεταβλητών **Ltarget** και **alpha**, όπου θα τα πάρουμε από την προηγούμενη γραμμή που υλοποιήσαμε, τα οποία είναι τα **alpha** και **L**.Στην προκειμένη για a=0.260 όπως και προηγουμένως. Επιπλέον αλλάζουμε και το directory στο MATLAB[1] .



Στην πορεία, αφού έχουμε εισάγει τα L και alpha ,τρέχουμε το αρχείο **alg0** καλώντας το στο terminal του MATLAB[1].Το **alg0** με την σειρά

του καλεί το **alg1**,το **alg2** το οποίο ολοκληρώνεται όταν φτάσει στο 501 και στην συνέχεια τα **alg3** και **alg4** τα οποία μας τυπώνουν τα δεδομένα που χρειαζόμαστε.

```
>> alg0
-------estimation reached---------

ans =

    20   501


ans =

    40   501
```

Αφού ολοκληρωθεί μας εμφανίζει δυο ομάδες αποτελεσμάτων. Η πρώτη ομάδα είναι τα αποτελέσματα τα οποία θα μπούνε στην 1η γραμμή κατά σειρά, πάνω από τα αποτελέσματα που βγάλαμε στο προηγούμενο κομμάτι του πειράματος, και η δεύτερη ομάδα στην 3η γραμμή,

Από την πρώτη ομάδα θα χρειαστούμε τα:

```
0.260007415889417   0.260007515016418   17.320996693582860   17.320997925896130   7.000000000000000

ans =

   4.500000000000000                   0   0.057768605419285   5.655372108385704
   4.259973671656970   0.038996050748545                   0   4.519947343313939
   4.500000000000000                   0   0.103983515277947   5.655372391977191
   4.259974864644376                   0   0.140706644937596   5.655372293383477
   4.500000000000000                   0   0.150198413278596   5.655372409835356
   4.259972566553124   0.077991769965937                   0   4.519945133106248
   4.500000000000000                   0   0.213743913012090   5.655372502768052
   4.259974194556305                   0   0.347628216638517   5.655372460183416
   4.259973647263824   2.768982870721485                   0   8.519947294527649
```

Το πρώτο σημειωμένο δεδομένο είναι το **alpha** μας, το δεύτερο είναι το **cost**,το τέταρτο είναι το **L** και το 5ο το **dC**,τα οποία και συμπληρώνουμε στο αρχείο με τα δεδομένα μας .Τα δεδομένα στις ενδιάμεσες στήλες, στην 2η και 3η στήλη, είναι τα **free** και **forced** δεδομένα που θα χρειαστούμε για τον υπολογισμό στο MAPLE[2].

```
ns =
                                free                                forced
  4.500000000000000                0   0.057768605419285   5.655372108385704
  4.259973671656970   0.038996050748545                   0   4.519947343313939
  4.500000000000000                0   0.103983515277947   5.655372391977191
  4.259974864644376                0   0.140706644937596   5.655372293383477
  4.500000000000000                0   0.150198413278596   5.655372409835356
  4.259972566553124   0.077991769965937                   0   4.519945133106248
  4.500000000000000                0   0.213743913012090   5.655372502768052
  4.259974194556305                0   0.347628216638517   5.655372460183416
  4.259973647263824   2.768982870721485                   0   8.519947294527649
```

Από την δεύτερη ομάδα θα χρειαστούμε τα:

```
ans =

  0.259974038437613   0.259974136956976   17.321156661365045   17.321157810231419   7.000000000000000


ans =

  4.500000000000000                0   0.057759918151266   5.655198363025317
  4.260092009646009   0.039013801446901                   0   4.520184019292017
  4.500000000000000                0   0.103967894622237   5.655198829135963
  4.260093213992733                0   0.140691899039036   5.655198812223956
  4.500000000000000                0   0.150175831451247   5.655198703471131
  4.260090928140667   0.078027278442200                   0   4.520181856281334
  4.500000000000000                0   0.213711799955876   5.655198918680410
  4.260092542601437                0   0.347591790912545   5.655199004345455
  4.260091978428756   2.769059785978691                   0   8.520183956857512
```

Αντίστοιχα όπως και στην πρώτη ομάδα, το πρώτο σημειωμένο δεδομένο είναι το **alpha** μας, το δεύτερο είναι το **cost**,το τέταρτο είναι το **dC** και το 5ο το **L**,τα οποία και συμπληρώνουμε στο αρχείο με τα δεδομένα μας.Τα δεδομένα στις ενδιάμεσες στήλες, στην 2η και 3η στήλη, είναι τα **free** και **forced** δεδομένα που θα χρειαστούμε για τον υπολογισμό στο MAPLE[2].

```
ns =
                                free                                forced
  4.500000000000000                0   0.057759918151266   5.655198363025317
  4.260092009646009   0.039013801446901                   0   4.520184019292017
  4.500000000000000                0   0.103967894622237   5.655198829135963
  4.260093213992733                0   0.140691899039036   5.655198812223956
  4.500000000000000                0   0.150175831451247   5.655198703471131
  4.260090928140667   0.078027278442200                   0   4.520181856281334
  4.500000000000000                0   0.213711799955876   5.655198918680410
  4.260092542601437                0   0.347591790912545   5.655199004345455
  4.260091978428756   2.769059785978691                   0   8.520183956857512
```

4.2.2 2η & 3η πειραματική γραμμή (MAPLE[2])

Σε αυτό το στάδιο έχουμε τελειώσει και με το MATLAB[1] και έχουμε ότι χρειαζόμαστε για να μεταφερθούμε στο αρχείο κώδικα του MAPLE[2],το οποίο είναι της μορφής:

```
d = 100

n = 8

a = [0.1, 0.15, 0.18, 0.17, 0.26, 0.3, 0.37, 0.42]

ad = a * d

A = sum of a from i = 1 to 8

Ad = sum of ad from i = 1 to 8

Ad = round down Ad to nearest integer

W = Ad

for i = 1 to n

   ad[i] = round down ad[i] to nearest integer

end for
```

```
p1 = 1 ,p2 = 0,p3 = 1,p4 = 0,p5 = 1,

p6 = 0,p7 = 1,p8 = 0

#Forced KS

result1 = p1 * ad[1] + p2 * ad[2] + p3 * ad[3] +
p4 * ad[4]

 + p5 * ad[5] + p6 * ad[6] + p7 * ad[7] + p8 *
ad[8]

#Free KS

result2 = (1 - p1) * ad[1] + (1 - p2) * ad[2] + (1 -
p3) * ad[3]
 + (1 - p4) * ad[4] + (1 - p5) * ad[5] + (1 - p6) *
ad[6] +
(1 - p7) * ad[7] + (1 - p8) * ad[8]
```

Η διαδικασία για την κάθε μία από τις δύο γραμμές είναι ακριβώς ίδια με το πρώτο κομμάτι του πειράματος, όπου επεξεργαζόμαστε τις τιμές που βρίσκονται κάτω από το σχόλιο #**forcedKS** ανάλογα με τα αποτελέσματα των **forced** και **free** που εξήγε το MATLAB[1] προηγουμένως. Όπου η ομάδα αριθμών forced έχει 0,στο MAPLE[2] βάζω 0 επίσης. Όπου έχει δεκαδικό αριθμό βάζω 1.

Δηλαδή για πρώτη ομάδα δεδομένων βάζουμε άσσους στα εξής:

p1 = 1

p3 = 1

p4 = 1

p5 = 1

p7 = 1

p8 = 1

και στα υπόλοιπα τοποθετούμε το 0 δηλαδή

p2 = 0

p6 = 0

Από κάτω παραθέτω ως παράδειγμα από την πρώτη ομάδα δεδομένων.

```
#forced KS
p1 := 1 : p2 := 0 : p3 := 1 : p4 := 1 : p5 := 1 : p6 := 0 : p7 := 1 : p8 := 1 :

p1 · ad[1] + p2 · ad[2] + p3 · ad[3] + p4 · ad[4] + p5 · ad[5] + p6 · ad[6] + p7 · ad[7] + p8 · ad[8];
#free KS

(1 − p1) · ad[1] + (1 − p2) · ad[2] + (1 − p3) · ad[3] + (1 − p4) · ad[4] + (1 − p5) · ad[5] + (1 − p6) · ad[6] + (1 − p7) · ad[7] + (1 − p8) · ad[8];

                                                                                                                    150
                                                                                                                    45
|
```

Αποθηκεύοντας και αυτά τα τελευταία δεδομένα στο συγκεντρωτικό αρχείο μας, έχει ολοκληρωθεί πλήρως μια πειραματική γραμμή και καθ' αυτόν τον τρόπο συνεχίζουμε μέχρις ότου να ολοκληρώσουμε τους αριθμούς μέσα στο διάστημα αυτό.

Το συγκεντρωτικό μας αρχείο είναι αυτής της μορφής:

```
-------------------------------------------------------------------------------
0.260007260167179{7}  {1,3,4,5,7,8}{2,6,9}[150-45]        17.320997514735488 5.655371503698106 4.259974242832969
0.260                 {1,2,3,5,7,8}{4,6,9}[148-47]        17.3248            5.67808            4.25595
0.259973880041386     {1,3,4,5,7,8}{2,6,9}[150-45]        17.321157446708177 5.655197832162313 4.260092589693357
-------------------------------------------------------------------------------
```

4.3 Συμπεράσματα

Με την ολοκλήρωση του 3ου πειράματος παιγνιοθεωρητικών στρατηγικών αρχηγού, εύκολα συμπεραίνει κανείς ορισμένα πράγματα από τη φύση του ίδιου του πειράματος. Κατ' αρχήν, για την ολοκλήρωση ενός πειράματος, από συγκεκριμένο εύρος τιμών που μελετήθηκαν, χρειάστηκαν 3 πειραματικές γραμμές από δεδομένα που εξορύξαμε και από τα 3 προγράμματα. Το γεγονός αυτό είχε ως αποτέλεσμα να πρέπει να συγκρίνουμε τις τιμές μεταξύ τους, κυρίως τα κόστη, και να καταγράφουμε τις περιπτώσεις οπού το κόστος της 2ης γραμμής υπερέβαινε το κόστος της 1ης ή της 3ης γραμμής. Σε μια τέτοια περίπτωση αποθηκεύαμε τον φάκελο **K**, τον οποίο παίρναμε σαν output από τη CLion[3], και τον αποστέλλαμε στον υπεύθυνο καθηγητή κ. Καπόρη για περεταίρω διερεύνηση.

Σχετικά με την εξαγωγή συμπερασμάτων από τα τρία προγράμματα:

- Στο **MATLAB[1]**, χρησιμοποιήσαμε για 1η φορά τον αλγόριθμο **alg0**, ο οποίος για να δεχθεί input τιμές

προαπαιτούσε να τρέξουμε τη γνωστή μας πλέον διαδικασία με τη συνάρτηση **stackoutputprint2**. Έπειτα, τοποθετώντας το **Ltarget** και το **alpha**, όπως προαναφέρθηκε, "έτρεχε" ο **alg0** για μερικά λεπτά, απαιτώντας μεγάλο τμήμα της CPU του υπολογιστή μας, και μας εμφάνιζε δεδομένα που μας ενδιέφεραν και τα οποία αποθηκεύαμε, αλλά και δεδομένα που τα χρησιμοποιήσαμε αργότερα σαν input στο MAPLE[2] (Forced-Free)

- Η συνάρτηση που χρησιμοποιήθηκε στο MAPLE[2], απαιτούσε και σε αυτό το πείραμα, δεδομένα τα οποία αντλούσαμε από τους φακέλους Forced & Free, και με το που "τρέχαμε" τις εντολές παίρναμε το αποτέλεσμα που επιθυμούσαμε άμεσα.
- Στην παρούσα πειραματική αξιολόγηση, χρειαστήκαμε τη CLion[3] ούτως ώστε να λάβουμε τους 5 φακέλους με δεδομένα, με τους τρόπους που εξηγήσαμε παραπάνω. Η εκτέλεση της εντολής **./prg L**, **dC**, **alpha** στο Ubuntu[4] ήταν για ακόμη μια φορά χρονοβόρα, μέχρι να εξάγει τα αποτελέσματα της.

Κεφάλαιο 5. Πείραμα 4

5.1.1. Προεργασία

Σχετικά με την προετοιμασία του τέταρτου πειράματος , μεταφέραμε από μέσω του remote pc,όπου είχαμε πρόσβαση στον H/Y του υπεύθυνου καθηγητή κ. Καπόρη, τα καινούρια αρχεία **HardKS** με τους υπόφακέλους hard1,hard2,..,hard10 από όπου τραβάει μέσω του **Fopen** δεδομένα το πρόγραμμα στην **CLion[3]**,το **outputHardRough8** από το οποίο τραβάει δεδομένα το MATLAB[1] με το νέο load_output.m, και το **Experiment-8_154_1000_10000-results.txt** στο οποίο θα συμπληρώνουμε γραμμή γραμμή τα δεδομένα από τις συγκρίσεις.

Για την λειτουργικότητα του MATLAB[1] στο νέο πείραμα, δημιουργούμε νέο φάκελο στον οποίο αντιγράφουμε από το remote PC το νέο outputHardRough8 και οδηγούμε το path του MATLAB[1] στο νέο αυτό φάκελο όπου υπάρχει μέσα το νέο **outputHardRough8**.Επίσης υπάρχει αρχείο κώδικα με όνομα **load_output_exp8.m** προσαρμοσμένο έτσι ώστε να μπορεί το MATLAB[1] να αντλήσει δεδομένα από το **outputHardRough8** και τέλος το αρχείο κώδικα με ονομασία **stackoutputprint2** τον οποίο χρησιμοποιούμε σε όλα τα πειράματα. Αφού ολοκληρωθεί αυτή η διαδικασία τρέχουμε την εντολή **load_output_exp8** και περιμένουμε να φορτωθούν τα δεδομένα από τους καταλόγους που υπάρχουν στο φάκελο, όπως ακριβώς και στο πείραμα 3. Με το πέρας της διαδικασίας και αφού έχουν γίνει όλα σωστά, το MATLAB[1] είναι έτοιμο να δεχθεί τις κατάλληλες εντολές που θα εξηγηθούν παρακάτω.

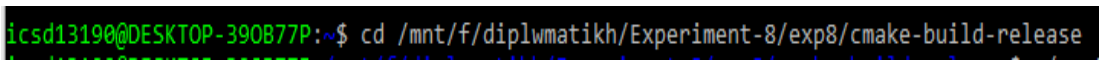
Τέλος, με την προεργασία του πειράματος, αντιγράφουμε το αρχείο κώδικα **forced-sum** της MAPLE[2] από το προηγούμενο πείραμα κατάλληλο για τον υπολογισμό δεδομένων κάθε πειραματικής γραμμής.

5.1.2 Αρχή πειράματος – 1η γραμμή (MATLAB[1]&Ubuntu[4])

Έχοντας αντικαταστήσει τα καινούρια **hardKS** ανοίγουμε τον κώδικα της CLion[3] και επιλέγουμε το "**rebuild all in Release**" έτσι ώστε να προσαρμοστεί το πρόγραμμα στα καινούρια δεδομένα.

Έπειτα είμαστε έτοιμοι να ανοίξουμε το Ubuntu[4]. Κάνουμε login με την εντολή **ssh icsd13190@localhost -p2222**(το όνομα χρήστη αλλάζει ανάλογα ποιός τρέχει το πείραμα)έτσι ώστε να συνδεθούμε στο **wls Toolchain**.Έπειτα μέσω της εντολής **cd** το οδηγούμε στο φάκελο

του υπολογιστή μας όπου έχουμε αποθηκευμένα αυτά τα **hardKS**, στην προκείμενη περίπτωση, στο φάκελο **cmake-build-release**, όπως φαίνεται παρακάτω.



Το συγκεκριμένο πείραμα θα αποτελείται από 2 γραμμές αποτελεσμάτων για κάθε **alpha**. Για την δημιουργία της πρώτης γραμμής αποτελεσμάτων η οποία βρίσκεται στο κέντρο ακολουθούμε την εξής διαδικασία.

Αφού έχουμε τρέξει την εντολή **load_output_Exp8** στο MATLAB[1] και τα δεδομένα μας έχουν φορτωθεί, το πρόγραμμα μας είναι έτοιμο και ξεκινάμε χρησιμοποιώντας τη συνάρτηση **[Vout, tt] = stackoutputprint2(x,x,costall)** , όπου x είναι τιμές στο κλειστό διάστημα [0,336 – 0,05]. Μελετάμε όλα τα **a** μέσα σε αυτό το διάστημα με βήμα 1/100 όπως φαίνεται στο αρχείο που είναι συγκεντρωμένα τα πειραματικά αποτελέσματα.

Θα αναλύσουμε ένα πείραμα εξηγώντας τη γενική μεθοδολογία που ακολουθήθηκε για την ολοκλήρωση κάθε μιας από τις πειραματικές γραμμές

Το παρακάτω παράδειγμα έχει εκτελεσθεί με τυχαίο αριθμό μέσα από το διάστημα, τον αριθμό a = 0.140.

Βρισκόμαστε στο περιβάλλον του MATLAB[1] και αρχικά πληκτρολογούμε την εντολή **format long** για το format των αριθμών. Έπειτα με το πάνω βελάκι του πληκτρολογίου μας αναζητούμε την εντολή **[Vout,tt]=stackoutputprint2(x,x,costall)** , εντολή η οποία καλεί την ομώνυμη συνάρτηση **stackoutputprint2**,όπου ως x βάζουμε το a = 0.140 και πατάμε το enter. Αμέσως μετά πληκτρολογούμε την εντολή **tt** στο command line και παίρνουμε το αποτέλεσμα που αφορά τον αριθμό 0,140.

```
>> load_output_Exp8
>> [Vout, tt]= stackoutputprint2(0.140 , 0.140 , costall);
>> tt

tt =

        3772
```

Τελειώνοντας, επιλέγουμε μαζί τις 3 εντολές

A2= Vout(1:tt,:);

A2= sortrows(A2,2);

A2(1,:);

για να λάβουμε το αποτέλεσμα που μας ενδιαφέρει, όπως φαίνεται στο παρακάτω στιγμιότυπο οθόνης.

```
>> A2= Vout(1:tt,:);
A2= sortrows(A2,2);
A2(1,:)

ans =

   1.0e+02 *
                          Cost                                   dC                  L
   0.001400000000000   0.175636000000000   0.027292100000000   0.057457600000000   0.044229000000000   5.680000000000000
```

Στην προκείμενη περίπτωση τα αποτελέσματα μας είναι διαιρεμένα με το 100, από bug του προγράμματος οπότε εμείς μετακινούμε την υποδιαστολή 2 θέσεις μέσα.

Από αυτή τη σειρά αποτελεσμάτων που εξάγει το MATLAB[1] θα χρησιμοποιήσουμε σαν input στο Ubuntu[4] για να μας δώσει αποτελέσματα η CLion[3], 2 παραμέτρους, το **dC** και το **L**.

Βασική προϋπόθεση, να πολλαπλασιάσουμε τους δυο αυτούς αριθμούς με το 100 πριν τους χρησιμοποιήσουμε στο άλλο πρόγραμμα. Τέλος, αποθηκεύουμε τις παραμέτρους **dC** και **L** μαζί με το **cost**, το οποίο αναγράφεται στην δεύτερη τιμή του παραπάνω screenshot.

Τελειώνοντας με το MATLAB[1] και έχοντας αντλήσει τα αποτελέσματα που χρειαζόμαστε για τη συνέχεια, μεταφερόμαστε στο Ubuntu[4], το οποίο, όπως αναφέραμε και προηγουμένως, έχουμε οδηγήσει στο φάκελο **cmake-build-release** του προγράμματος και είναι έτοιμο να δεχθεί το input ώστε να τρέξει και να εξάγει αποτελέσματα η CLion[3].

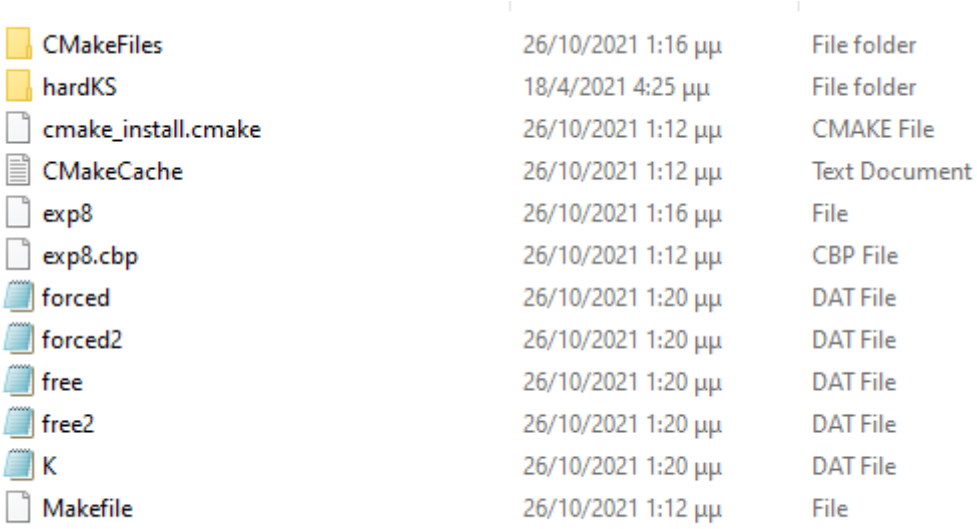
Σε αυτό το σημείο εκτελούμε στο Ubuntu[4] την παρακάτω εντολή:

```
icsd13190@DESKTOP-39OB77P:/mnt/f/diplwmatikh/Experiment-8/exp8/cmake-build-release$ ./exp8 4.422900 5.745760 0.140
 inst = ,       9,      4.4229, 4.4229, 1.43051e-06,    5.74576,        5.74576,        1.43051e-06,    0.14
```

Αριστερά με τα μπλε γράμματα αναγράφεται το path που οδηγεί στο build release του προγράμματος μας, και παραμένει σταθερό για όλη τη διάρκεια της εκτέλεσης των πειραμάτων. Έπειτα, ακολουθεί η εντολή **./exp8** , όπου exp8 το όνομα που δόθηκε στο Project, και έπειτα τρεις αριθμοί. Αυτοί οι αριθμοί με τη σειρά είναι : πρώτα το **L** από το MATLAB[1] έπειτα το **dC** από το ίδιο πρόγραμμα και τέλος το εκάστοτε **alpha** που τρέχουμε κάθε φορά, στην προκείμενη περίπτωση το 0,140.

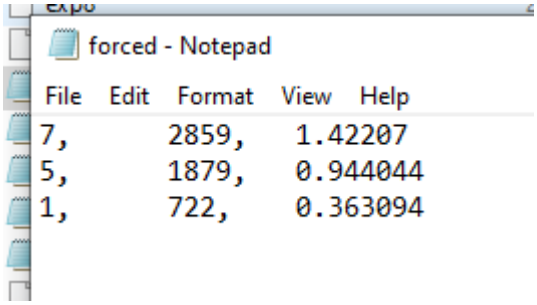
Πατώντας το enter αρχίζει να "τρέχει" ο αλγόριθμος και περιμένουμε λίγα λεπτά ούτος ώστε να πάρουμε τα αποτελέσματα του.

Αφού ολοκληρωθεί η διαδικασία ο κώδικας εξάγει σαν αποτελέσματα πέντε αρχεία στον φάκελο που τον έχουμε οδηγήσει.

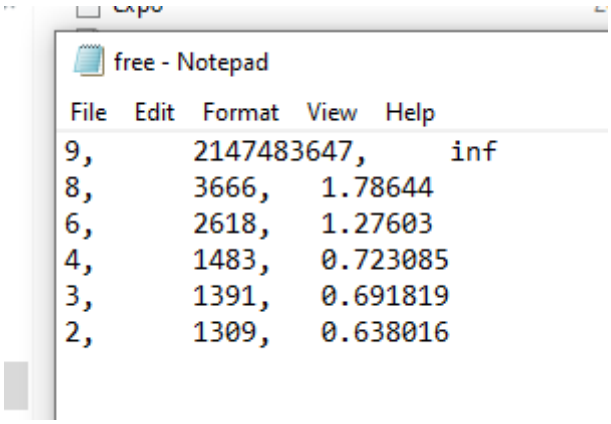


Πρόκειται για τα αρχεία **forced** και **free** τα οποία χρησιμοποιούμε αργότερα στον κώδικα του MAPLE[2],όπως και για το αρχείο K

Τα αρχεία **forced** και **free** εμπεριέχουν αριθμούς από το 1 μέχρι το 9, για παράδειγμα στον αριθμό που εξετάζουμε, το αρχείο **forced** έβγαλε το συγκεκριμένο αποτέλεσμα,



Οπότε αναμένουμε από το αρχείο **free** να έχει τους αριθμούς από το διάστημα [1,9] που δεν αναγράφονται στο αρχείο **forced**. Πράγματι όπως φαίνεται παρακάτω αυτό συμβαίνει:



5.1.3 1η Πειραματική γραμμή – Συνέχεια (MAPLE[2])

Σε αυτό το στάδιο έχουμε τελειώσει και με το **Ubuntu[4] – CLion[3]** και έχουμε ότι χρειαζόμαστε για να μεταφερθούμε στο 3ο και τελευταίο στάδιο της πρώτης γραμμής των πειραμάτων, στο αρχείο κώδικα του **MAPLE[2]**.

Ο κώδικας αυτός είναι της μορφής που φαίνεται παρακάτω:

```
d = 100

n = 8

a = [0.1, 0.15, 0.18, 0.17, 0.26, 0.3, 0.37, 0.42]

ad = a * d

A = sum of a from i = 1 to 8

Ad = sum of ad from i = 1 to 8

Ad = round down Ad to nearest integer

W = Ad

for i = 1 to n

   ad[i] = round down ad[i] to nearest integer

end for
```

```
p1 = 1 ,p2 = 0,p3 = 1,p4 = 0,p5 = 1,

p6 = 0,p7 = 1,p8 = 0

#Forced KS

result1 = p1 * ad[1] + p2 * ad[2] + p3 * ad[3] +
p4 * ad[4]

 + p5 * ad[5] + p6 * ad[6] + p7 * ad[7] + p8 *
ad[8]

#Free KS

result2 = (1 - p1) * ad[1] + (1 - p2) * ad[2] + (1 -
p3) * ad[3]
 + (1 - p4) * ad[4] + (1 - p5) * ad[5] + (1 - p6) *
ad[6] +
(1 - p7) * ad[7] + (1 - p8) * ad[8]
```

Το κομμάτι του κώδικα MAPLE[2] που επεξεργαζόμαστε σε κάθε πείραμα είναι οι τιμές που βρίσκονται κάτω από το σχόλιο #**forcedKS**.

Εκεί υπάρχουν οκτώ pi, οπού i οι ακέραιοι αριθμοί που ανήκουν στο διάστημα [1,8], και δέχονται μόνο τιμές 0 και 1.

Για να καθοριστεί το τι τιμές θα πάρουν αυτά τα p1,p2,…,p8, δηλαδή αν θα έχουν 0 η 1, αρκεί να εξετάσουμε το αρχείο **forced** που εξάγει η CLion[3] και να βάλουμε άσσους στα αντίστοιχα pi των αριθμών που αναγράφονται στο αρχείο.

Δηλαδή στην προκείμενη περίπτωση βάζουμε άσσους στα εξής:

p1 = 1

p5 = 1

p7 = 1

και στα υπόλοιπα τοποθετούμε το 0 δηλαδή

p2 = 0

p3 = 0

p4 = 0

p6 = 0

p8 = 0

Αφού συμπληρώσουμε τις τιμές καθ’ αυτόν τον τρόπο είμαστε έτοιμοι να πατήσουμε το enter και ο αλγόριθμος να δουλέψει ελάχιστα

δευτερόλεπτα και να πάρουμε το αποτέλεσμα που χρειαζόμαστε το οποίο είναι το 73,122.

```
#forced KS
p1 := 1 : p2 := 0 : p3 := 0 : p4 := 0 : p5 := 1 : p6 := 0 : p7 := 1 : p8 := 0 :

p1·ad[1] + p2·ad[2] + p3·ad[3] + p4·ad[4] + p5·ad[5] + p6·ad[6] + p7·ad[7] + p8·ad[8];
#free KS

(1 − p1)·ad[1] + (1 − p2)·ad[2] + (1 − p3)·ad[3] + (1 − p4)·ad[4] + (1 − p5)·ad[5] + (1 − p6)·ad[6] + (1 − p7)·ad[7] + (1 − p8)·ad[8];

                                                                    73
                                                                    122
```

Οι δυο αυτοί αριθμοί ,73 και 122 στην προκείμενη περίπτωση, και γενικά σε όλα τα πειράματα πρέπει να πούμε πως θα έχουν πάντα άθροισμα το 195 και είναι ένας τρόπος να επαληθεύσουμε ότι έχουμε κάνει σωστά τη διαδικασία.

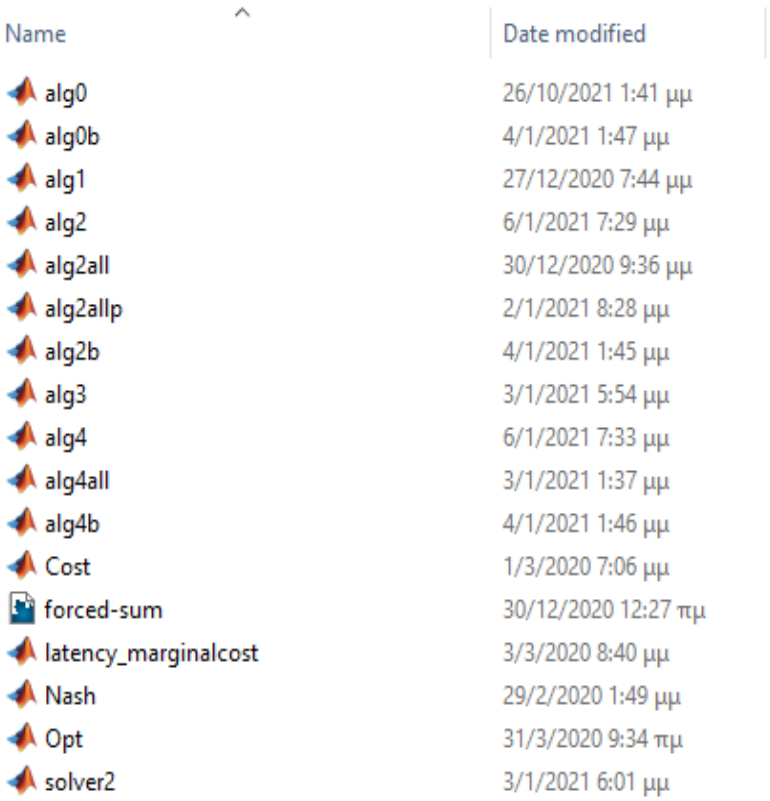
Αποθηκεύοντας και αυτά τα τελευταία δεδομένα στο συγκεντρωτικό αρχείο μας, έχει ολοκληρωθεί πλήρως την πρώτη πειραματική γραμμή και καθ' αυτόν τον τρόπο συνεχίζουμε μέχρις ότου να ολοκληρώσουμε τους αριθμούς μέσα στο διάστημα αυτό.

```
0.141                 {1,3,5,7}{2,4,6,8,9}[91-104] 17.55750          5.448520          4.445680
```

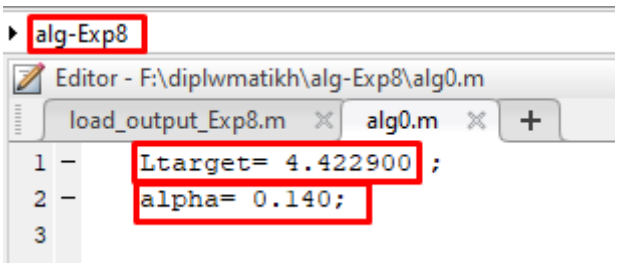
Αφού ολοκληρώσαμε την 1η γραμμή δεδομένων του πειράματος, συνεχίζουμε ώστε να συμπληρώσουμε και την επόμενη γραμμή.

5.2.1 2η Πειραματική γραμμή μέσω alg0 (MATLAB[1])

Στην προκειμένη περίπτωση θα χρησιμοποιήσουμε μόνο τα προγράμματα MATLAB[1] και MAPLE[2].Δημιουργούμε ένα νέο φάκελο με το όνομα alg-Exp8 στον οποίο μεταφέρουμε από το υπολογιστή του κ Καπόρη μέσω του remote PC ένα σύνολο αρχείων προγραμμάτων MATLAB[1] όπως και το forced-sum που χρησιμοποιήσαμε προηγουμένως.



Αρχικά ανοίγουμε το αρχείο **alg0.m** μέσω του MATLAB[1] στο οποίο θα εισάγουμε τα δεδομένα με την επεξεργασία των δύο πρώτων μεταβλητών **Ltarget** και **alpha**, όπου θα τα πάρουμε από την προηγούμενη γραμμή που υλοποιήσαμε, τα οποία είναι τα **alpha** και **L**.Στην προκειμένη για **a=0.140** όπως και προηγουμένως. Επιπλέον αλλάζουμε και το directory στο MATLAB[1] .



Στην πορεία, αφού έχουμε εισάγει τα **L** και **alpha** ,τρέχουμε το αρχείο **alg0** καλώντας το στο terminal του MATLAB[1].Το **alg0** με την σειρά του καλεί το **alg1**,το **alg2** το οποίο ολοκληρώνεται όταν φτάσει στο 251

και στην συνέχεια τα **alg3** και **alg4** τα οποία μας τυπώνουν τα δεδομένα που χρειαζόμαστε.

```
>> alg0
-------estimation reached---------

ans =

    20   251


ans =

    40   251
```

Αφού ολοκληρωθεί μας εμφανίζει δυο ομάδες αποτελεσμάτων. Η πρώτη ομάδα είναι τα αποτελέσματα τα οποία θα μπούνε στην 1η γραμμή κατά σειρά, πάνω από τα αποτελέσματα που βγάλαμε στο προηγούμενο κομμάτι του πειράματος, και είναι και αυτή η οποία θα χρησιμοποιήσουμε, αντιθέτως με το προηγούμενο πείραμα που χρειαστήκαμε και τις δύο.

Από την πρώτη ομάδα θα χρειαστούμε τα:

```
ans =

  0.140009576270907   0.140009528666562  17.560961807242727  17.560961959239560  18.000000000000000


ans =

   4.422440123532659                   0   0.072265359470618   5.745307189412364
   4.422440048001100   0.063366007200165                   0   4.844880096002200
   4.422439723529235   0.040039150235262                   0   4.644879447058471
   4.422440044380735   0.071814807544725                   0   4.844880088761472
   4.422440056779316                   0   0.187889978925850   5.745307530198843
   4.422439707329548   0.126731912198865                   0   4.844879414659097
   4.422440483711084                   0   0.285881823403123   5.745307153530392
   4.422439746654421   0.177424693594857                   0   4.844879493308841
   4.422440411425439   2.874586267426535                   0   8.844880822850879
```

Το πρώτο σημειωμένο δεδομένο είναι το **alpha** μας, το δεύτερο είναι το **cost**,το τέταρτο είναι το **L** και το 5ο το **dC**,τα οποία και συμπληρώνουμε στο αρχείο με τα δεδομένα μας .Τα δεδομένα στις ενδιάμεσες στήλες, στην 2η και 3η στήλη, είναι τα **free** και **forced** δεδομένα που θα χρειαστούμε για τον υπολογισμό στο MAPLE[2].

	free	forced	
4.422440123532659	0	0.072265359470618	5.745307189412364
4.422440048001100	0.063366007200165	0	4.844880096002200
4.422439723529235	0.040039150235262	0	4.644879447058471
4.422440044380735	0.071814807544725	0	4.844880088761472
4.422440056779316	0	0.187889978925850	5.745307530198843
4.422439707329548	0.126731912198865	0	4.844879414659097
4.422440483711084	0	0.285881823403123	5.745307153530392
4.422439746654421	0.177424693594857	0	4.844879493308841
4.422440411425439	2.874586267426535	0	8.844880822850879

5.2.2 2η Πειραματική γραμμή – Επιστροφή στο MAPLE[2]

Σε αυτό το στάδιο έχουμε τελειώσει και με το MATLAB[1] και έχουμε ότι χρειαζόμαστε για να μεταφερθούμε στο αρχείο κώδικα του MAPLE[2],το οποίο σε ψευδοκώδικα είναι της μορφής:

```
d = 100

n = 8

a = [0.1, 0.15, 0.18, 0.17, 0.26, 0.3, 0.37, 0.42]

ad = a * d

A = sum of a from i = 1 to 8

Ad = sum of ad from i = 1 to 8

Ad = round down Ad to nearest integer

W = Ad

for i = 1 to n

   ad[i] = round down ad[i] to nearest integer

end for
```

```
p1 = 1 ,p2 = 0,p3 = 1,p4 = 0,p5 = 1,

p6 = 0,p7 = 1,p8 = 0

#Forced KS

result1 = p1 * ad[1] + p2 * ad[2] + p3 * ad[3] +
p4 * ad[4]

 + p5 * ad[5] + p6 * ad[6] + p7 * ad[7] + p8 *
ad[8]

#Free KS

result2 = (1 - p1) * ad[1] + (1 - p2) * ad[2] + (1 -
p3) * ad[3]
 + (1 - p4) * ad[4] + (1 - p5) * ad[5] + (1 - p6) *
ad[6] +
(1 - p7) * ad[7] + (1 - p8) * ad[8]
```

Η διαδικασία για την κάθε μία από τις δύο γραμμές είναι ακριβώς ίδια με το πρώτο κομμάτι του πειράματος, όπου επεξεργαζόμαστε τις

τιμές που βρίσκονται κάτω από το σχόλιο #**forcedKS** ανάλογα με τα αποτελέσματα των **forced** και **free** που εξήγε το MATLAB[1] προηγουμένως. Όπου η ομάδα αριθμών **forced** έχει 0,στο MAPLE[2] βάζω 0 επίσης. Όπου έχει δεκαδικό αριθμό βάζω 1.

Δηλαδή για πρώτη ομάδα δεδομένων βάζουμε άσσους στα εξής:

p1 = 1

p5 = 1

p7 = 1

και στα υπόλοιπα τοποθετούμε το 0 δηλαδή

p2 = 0

p3 = 0

p4 = 0

p6 = 0

p8 = 0

Από κάτω παραθέτω ως παράδειγμα από την πρώτη ομάδα δεδομένων.

```
#forced KS
p1 := 1 : p2 := 0 : p3 := 0 : p4 := 0 : p5 := 1 : p6 := 0 : p7 := 1 : p8 := 0 :

p1·ad[1] + p2·ad[2] + p3·ad[3] + p4·ad[4] + p5·ad[5] + p6·ad[6] + p7·ad[7] + p8·ad[8];
#free KS

(1 − p1)·ad[1] + (1 − p2)·ad[2] + (1 − p3)·ad[3] + (1 − p4)·ad[4] + (1 − p5)·ad[5] + (1 − p6)·ad[6] + (1 − p7)·ad[7] + (1 − p8)·ad[8];

                                                                    73
                                                                   122
```

Αποθηκεύοντας και αυτά τα τελευταία δεδομένα στο συγκεντρωτικό αρχείο μας, έχει ολοκληρωθεί πλήρως μια πειραματική γραμμή και καθ' αυτόν τον τρόπο συνεχίζουμε μέχρις ότου να ολοκληρώσουμε τους αριθμούς μέσα στο διάστημα αυτό.

Το συγκεντρωτικό μας αρχείο είναι αυτής της μορφής:

```
-------------------------------------------------------------------------------
0.141004728535044(16) {1,3,5,7}{2,4,6,8,9}[91-104] 17.550399012522533 5.448171626939493 4.443835511240175
0.141                 {1,3,5,7}{2,4,6,8,9}[91-104] 17.55750           5.448520          4.445680
-------------------------------------------------------------------------------
```

5.3 Συμπεράσματα

Το πέρας του 4ου πειράματος παιγνιοθεωρητικών στρατηγικών αρχηγού, μας οδηγεί στο να επισημάνουμε τις ομοιότητες του με το 3ο μας πείραμα, τόσο στη μεθοδολογία όσο και στην εκτέλεση τους. Αρχικά, για την ολοκλήρωση ενός πειράματος, τα **alpha** που μελετήθηκαν ανήκαν και αυτά σε συγκεκριμένο εύρος τιμών, ενώ, χρειάστηκαν 2 πειραματικές γραμμές από δεδομένα που εξορύξαμε και από τα 3 προγράμματα, αυτή τη φορά. Και σε αυτήν την περίπτωση, λοιπόν, θα πρέπει να συγκρίνουμε τις τιμές μεταξύ τους, κυρίως τα **κόστη**, αλλά και τα **alpha** και να καταγράφουμε τις περιπτώσεις οπού το κόστος της 1ης γραμμής υπερέβαινε το κόστος της 2ης γραμμής. Σε μια τέτοια περίπτωση αποθηκεύαμε τον φάκελο **K**, τον οποίο παίρναμε σαν output από τη CLion[3], και τον αποστέλλαμε στον υπεύθυνο καθηγητή κ. Καπόρη για περεταίρω διερεύνηση.

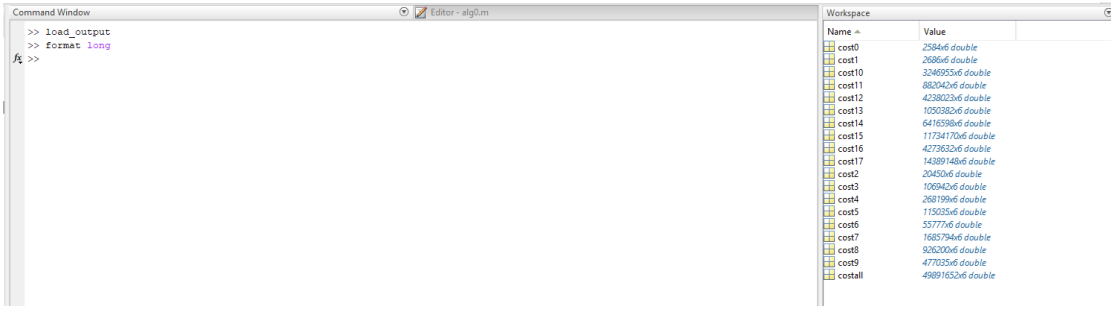
Σχετικά με τις ομοιότητες στην εξαγωγή συμπερασμάτων από τα τρία προγράμματα:

- Στο **MATLAB[1]**, χρησιμοποιήσαμε και εδώ τον αλγόριθμο **alg0**, προσαρμοσμένο στο νέο πείραμα, ο οποίος για να δεχθεί input τιμές προαπαιτεί να «τρέξουμε» τη διαδικασία με τη συνάρτηση **stackoutputprint2**. Έπειτα, τοποθετώντας το **Ltarget** και το **alpha**, όπως προαναφέρθηκε, "έτρεχε" ο **alg0** για μερικά λεπτά, απαιτώντας μεγάλο τμήμα της CPU του υπολογιστή μας, και μας εμφάνιζε δεδομένα που μας ενδιέφεραν και τα οποία αποθηκεύαμε, αλλά και δεδομένα που τα χρησιμοποιήσαμε αργότερα σαν input στο MAPLE[2] (Forced-Free)
- Η συνάρτηση και η μεθοδολογία που χρησιμοποιήθηκε στο **MAPLE[2]**, για το παρών πείραμα 5 παραμένουν ιδιά, δηλαδή τα δεδομένα τα οποία αντλούσαμε από τους φακέλους Forced & Free, τα προσαρμόζουμε στην συνάρτηση και παίρνουμε το αποτέλεσμα που επιθυμούμε άμεσα. Θα πρέπει να αναφέρουμε επίσης, πως έγινε χρήση της συνάρτησης του MAPLE[2] και στις 2 γραμμές του ίδιου πειράματος.
- Ιδιά η μεθοδολογία και για τη **CLion[3]**, καθώς λαμβάνουμε τους 5 φακέλους με δεδομένα, με τους τρόπους που εξηγήσαμε παραπάνω. Η εκτέλεση της εντολής **./exp8 L**, **dC**, **alpha** στο Ubuntu[4] ήταν και τώρα χρονοβόρα, μέχρι να εξάγει τα αποτελέσματα της.

Κεφάλαιο 6. Πείραμα 5

6.1 Προεργασία

Σχετικά με την προετοιμασία του πέμπτου πειράματος, μεταφέραμε μέσω του remote pc,όπου είχαμε πρόσβαση στον H/Y του υπεύθυνου καθηγητή κ. Καπόρη, τον φάκελο **outputHardRough10** που περιέχει το νέο **load_output.m** του MATLAB[1], τον φάκελο out10 από το οποίο αντλεί τα δεδομένα που θα φορτώσει το load_output. Καταλαβαίνουμε πως έχει ολοκληρωθεί η διαδικασία όταν στο workspace του προγράμματος έχουν φορτωθεί τα ανάλογα κόστη όπως φαίνεται παρακάτω



Επίσης, το **KS-VS-Alg0-exp10.txt** στο οποίο θα συμπληρώνουμε γραμμή τα δεδομένα από τις συγκρίσεις, το αρχείο κώδικα **knapsack_D2_part1234** του MATLAB[1], όπου γεμίζει τους υποφακέλους **KS1, KS2, KS3 , KS4** με 2 αρχεία **(**data.txt, data2.txt**)**, τα οποία επίσης μεταφέραμε, όπως και τα 4 αρχεία knapsack_D2_part1 έως 4 τα οποία καλούνται με την κλίσης του **knapsack_D2_part1234** , 4 αρχεία MAPLE[2] **knapsack-1dimension-MATLAB[1]-input_10_knapsack_D_part1,2,3,4** τα οποία διαβάζουν δεδομένα από τους υποφακέλους **KS1, KS2, KS3 , KS4,**το αρχείο κώδικα MAPLE[2] alg0.m το οποίο διαμορφώνουμε, όπως επίσης και κάποια άλλα αρχεία MAPLE[2] που θα χρειαστούν για το πείραμά μας.Τέλος μεταφέραμε και τον φάκελο **inputHardRough10** με 10 υποφακέλους hard1 έως 10.

Για την λειτουργικότητα του MATLAB[1] στο νέο πείραμα, δημιουργούμε νέο φάκελο experiment-10 στον οποίο αντιγράφουμε από το remote PC το νέο **outputHardRough10** και **inputHardRough10** και οδηγούμε το path του MATLAB[1] στο φάκελο **outputHardRough10**.Επίσης υπάρχει αρχείο κώδικα με όνομα

stackoutputprint2 τον οποίο χρησιμοποιούμε σε όλα τα πειράματα. Αφού ολοκληρωθεί αυτή η διαδικασία τρέχουμε την εντολή **load_output** και περιμένουμε να φορτωθούν τα δεδομένα από τους καταλόγους που υπάρχουν στο φάκελο, όπως ακριβώς και στο πείραμα 4. Με το πέρας της διαδικασίας και αφού έχουν γίνει όλα σωστά, το MATLAB[1] είναι έτοιμο να δεχθεί τις κατάλληλες εντολές που θα εξηγηθούν παρακάτω.

6.2.1 Έναρξη πειραματικής διαδικασίας (MATLAB[1])

Για την έναρξη της διαδικασίας, ο υπεύθυνος καθηγητής κ. Καπόρης, μας όρισε το διάστημα όπου λαμβάνονται οι μετρήσεις στο [3.0 – 3.9] και με βήμα 1/100 ωστόσο επισημάνθηκε πως έπρεπε να εξεταστούν και ενδιάμεσες τιμές για να εξετάσουμε τη συμπεριφορά τους. Για παράδειγμα αναμεσά στους αριθμούς 3.01 και 3.02 θα πρέπει να εξετάσουμε επιπρόσθετα τους αριθμούς 3.013 και 3.016 που βρίσκονται αναμεσά έτσι ώστε να λάβουμε ασφαλέστερα και σωστότερα αποτελέσματα.

Για να εξηγήσουμε την πειραματική διαδικασία επιλέξαμε τυχαία τον αριθμό 3.86 που βρίσκεται μέσα σε αυτό το διάστημα.

Αρχικά μεταφερόμαστε στο περιβάλλον του MATLAB[1] και παράλληλα με το command line του προγράμματος «ανοίγουμε» και το κομμάτι με τον ένθετο κώδικα που έχουμε ονομάσει alg0 και εκεί στο κομμάτι του κώδικα που ονομάζεται Ltarget πάμε και αλλάζουμε κάθε φορά την τιμή και προσθέτουμε οποία εξετάζουμε. Στην προκειμένη περίπτωση το alg0 σε ψευδοκώδικα είναι της παρακάτω μορφής.

```
INITIALIZE Ltarget to 3.86

INITIALIZE SS to 4

INITIALIZE delta to 1/500

INITIALIZE width to 0

INITIALIZE error1 to 0.000001

INITIALIZE eps to 0.0001

SET format to long

INITIALIZE An to [0.12 0.1 0.17 0.13 0.3 0.15 0.4 0.2]

CALCULATE A_total as the sum of elements in An

CALCULATE a as [1/An[1], 1/An[2], 1/An[3], 1/An[4], 1/An[5],
1/An[6], 1/An[7], 1/An[8], 3/A_total[1]]

INITIALIZE b as [3 + 1.6, 3 + 1.2, 3 + 0.8, 3 + 0.5, 3 + 0.3, 3 + 0.2, 3 +
0.1, 3 + 0.0, 0]

CALCULATE n as the length of a

CALCULATE r_total as 2 * A_total
CALL alg1
CALL alg2d1
```

Σε αυτό το σημείο μεταφερόμαστε πίσω στο command line του προγράμματος και πληκτρολογούμε την εντολή alg0 η οποία τρέχει τον ένθετο κώδικα alg0, και άμεσα λαμβάνουμε από το MATLAB[1] τα εξής αποτελέσματα

```
>> alg0
-------estimation reached---------
alpha            Rtarget

ans =

   0.101888656616211   2.820069618225098

LatencyBound          LatencyBound2             minD             maxD

ans =

   5.388668060302735   5.388668060302735   3.732930320570067   7.326639827937418

>>
```

Από τα παραπάνω αποτελέσματα του alg0, η σημαντική παράμετρος είναι η παρακάτω :

```
>> alg0
-------estimation reached---------
alpha            Rtarget

ans =

   0.101888656616211   2.820069618225098

LatencyBound          LatencyBound2             minD             maxD

ans =

   5.388668060302735   5.388668060302735   3.732930320570067   7.326639827937418

>>
```

τιμή την οποία χρησιμοποιεί αργότερα η stackoutputprint2 αποθηκευμένο στη μεταβλητή alpha.

Αμέσως μετα, εκτελούμε στο command line την εξής εντολή :

```
>> knapsack_D2_part1234
>>
```

knapsack_D2_part1234 η οποία, αφού τρέξει για μερικά δευτερόλεπτα, εντοπίζει τους φακέλους KS1, KS2, KS3 & KS4 και τους "γεμίζει" με δυο φακέλους, τους data & data2 .txt, όπως βλέπουμε παρακάτω:

Όνομα	Ημερομηνία τροποποι...	Τύπος	Μέγεθος
data.txt	27/10/2021 4:24 μμ	Έγγραφο κειμένου	1.453 KB
data2.txt	27/10/2021 4:24 μμ	Έγγραφο κειμένου	279 KB

6.2.2 Χρήση τεσσάρων νέων αρχείων κώδικα (MAPLE[2])

Τα δεδομένα αυτά διαβάζονται κατ' αντιστοιχία από τα τέσσερα αρχεία Maple που έχουμε αντιγράψει στον υπολογιστή μας,

- knapsack-1dimension-MATLAB[1]-input_10_knapsack_D_part1,
- knapsack-1dimension-MATLAB[1]-input_10_knapsack_D_part2,
- knapsack-1dimension-MATLAB[1]-input_10_knapsack_D_part3,
- knapsack-1dimension-MATLAB[1]-input_10_knapsack_D_part4

Το αρχείο knapsack-1dimension-MATLAB[1] input_10_knapsack_D_part1 αντλεί δεδομένα από τον KS1, το αρχείο knapsack-1dimension-MATLAB[1]-input_10_knapsack_D_part2 αντλεί

δεδομένα από τον KS2 και ούτω καθεξής. Θα πρέπει σε αυτό το σημείο να επισημάνουμε πως μετα το πέρας ενός πειράματος είμαστε υποχρεωμένοι να σβήνουμε τα αρχεία αυτά που αφορούσαν το προηγούμενο Ltarget ώστε, ακολουθώντας την ιδιά διαδικασία, να «γεμίσουν» και πάλι.

Μόλις λοιπόν το MATLAB[1] «γεμίσει» με δεδομένα τα δυο αυτά αρχεία και στους τέσσερις φακέλους είμαστε έτοιμοι να μεταφερθούμε στα 4 αρχεία MAPLE[2] που προαναφέραμε. Για την εκτέλεση αυτών αρκεί να μεταφέρουμε τον κέρσορα και να πατήσουμε ένα enter στην αρχή του καθενός, ακριβώς δίπλα στο restart

```
> restart :
  d := 10000 :
  part := 1 :

  data := readdata("./KS1/data.txt", 5) :
  data2 := readdata("./KS1/data2.txt", 8) :
```

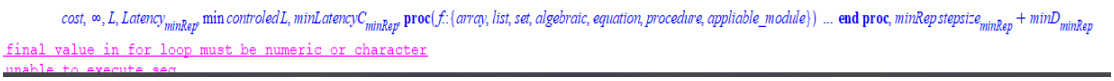
Τότε ξεκινά η εκτέλεση των εντολών και το κάθε ένα αρχείο είναι προγραμματισμένο να εκτελεί 2500 επαναλήψεις. Αφού το πρόγραμμα μας τρέχει ταυτόχρονα και τα 4 parts καταλαβαίνουμε ότι έχει τελειώσει η διαδικασία όταν ολοκληρωθούν οι 2500 επαναλήψεις και δούμε το παρακάτω αποτέλεσμα

```
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2230, 4.022157778, 5.388668060, 4.366197754, 1.396569072, 12.73933402
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3199, 2231, 4.022157778, 5.388668060, 4.366377439, 1.396861918, 12.73962687
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3199, 2232, 4.022157778, 5.388668060, 4.366557125, 1.397154784, 12.73991973
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3194, 2299, 3.914384232, 5.388668060, 4.478596052, 1.480374294, 12.51921034
                                                                "rep", 2300
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3194, 2300, 3.914384232, 5.388668060, 4.478775737, 1.480786150, 12.51962220
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3195, 2301, 3.914384232, 5.388668060, 4.478955422, 1.481198034, 12.52003408
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3195, 2302, 3.914384232, 5.388668060, 4.479135108, 1.481609944, 12.52044599
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3197, 2303, 3.914384232, 5.388668060, 4.479314794, 1.482021882, 12.52085793
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2304, 3.914384232, 5.388668060, 4.479494479, 1.482433847, 12.52126990
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2305, 3.914384232, 5.388668060, 4.479674165, 1.482845839, 12.52168189
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2306, 3.914384232, 5.388668060, 4.479853850, 1.483257858, 12.52209391
                                                                "rep", 2400
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3194, 2461, 3.990828730, 5.388668060, 4.357705099, 1.435879972, 12.69029482
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3194, 2462, 3.990828730, 5.388668060, 4.357884784, 1.436197114, 12.69061196
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3195, 2463, 3.990828730, 5.388668060, 4.358064469, 1.436514278, 12.69092913
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3196, 2464, 3.990828730, 5.388668060, 4.358244155, 1.436831462, 12.69124631
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3196, 2465, 3.990828730, 5.388668060, 4.358423841, 1.437148667, 12.69156352
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3196, 2466, 3.990828730, 5.388668060, 4.358603526, 1.437465892, 12.69188074
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3197, 2467, 3.990828730, 5.388668060, 4.358783211, 1.437783139, 12.69219799
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2468, 3.990828730, 5.388668060, 4.358962897, 1.438100406, 12.69251526
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2469, 3.990828730, 5.388668060, 4.359142583, 1.438417693, 12.69283254
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2470, 3.990828730, 5.388668060, 4.359322268, 1.438735001, 12.69314985
                                                                "rep", 2500
, Latency[minRep],'min controledL', minLatencyC[minRep], 'D', minD[minRep] + (part - 1) · 2500 · stepsize[minRep] + minRep · (stepsize[minRep]));
```

Σε αυτό το σημείο το κάθε part είναι έτοιμο να μας δώσει αποτελέσματα αρκεί να μεταφέρουμε τον κέρσορα και να πατήσουμε enter ακριβώς στις από κάτω εντολές που είναι ιδίες και στα 4 part.

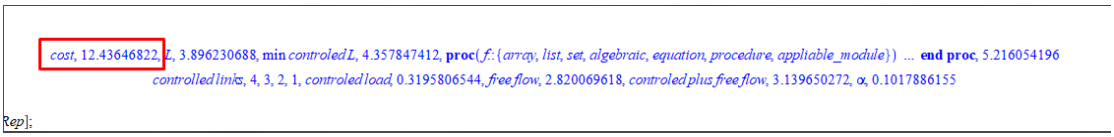
```
controledload := 0 :
for s from 1 by 1 to tS[minRep] do

controledload := controledload + wf[minRep, S[minRep, s]] :

end do:
```

Το 1ο part στο παράδειγμα μας έβγαλε τα εξής αποτελέσματα



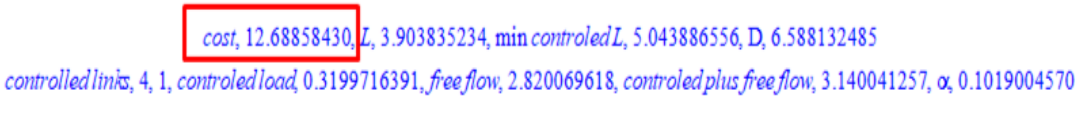
δηλαδή παρουσίασε σφάλμα κατά την διαδικασία των επαναλήψεων γεγονός που δεν μας προβληματίζει καθώς θα εξάγουμε τα αποτελέσματα μας από τα υπόλοιπα τρία parts και θα επιλέξουμε τις τιμές από το part που παρουσίασε το **μικρότερο κόστος.**

Έπειτα εξετάζουμε το 2ο part με κόστος 12,43646822



Μετέπειτα εξετάζουμε το 3ο part με κόστος 12,47818118

Τέλος εξετάζουμε το 4ο και τελευταίο part με κόστος 12,68858430



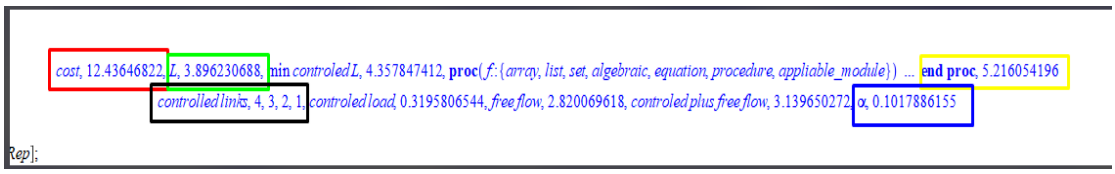
6.3 Σύγκριση αποτελεσμάτων και συνέχεια της διαδικασίας (επιστροφή στο MATLAB[1])

Συγκρίνοντας και τα 4 parts καταλήγουμε σε αυτό με το μικρότερο κόστος, στην προκείμενη περίπτωση επιλέγουμε το part2 και ακολουθούμε την εξής διαδικασία:

από το σύνολο των παραμέτρων που εμφανίζονται επιλέγουμε και αποθηκεύουμε τα εξής:

- **a** την τελευταία τιμή (μπλε χρώμα)
- **cost**, όπως προαναφέρθηκε, η 1η τιμή (κόκκινο χρώμα)
- σαν **L** επιλέγουμε την 2η παράμετρο(πράσινο χρώμα)
- σαν **D** επιλέγουμε την τελευταία παράμετρο της 1ης σειράς (κίτρινο χρώμα)
- Τέλος, από τα παραπάνω αποτελέσματα αποθηκεύουμε στο συγκεντρωτικό αρχείο txt αυτού του πειράματος τα **controlled links** (μαύρο χρώμα)

Στο παρακάτω screenshot φαίνονται οι τιμές που μας απασχολούν με τα παραπάνω χρώματα για να είναι κατανοητή η διαδικασία



Αντιγράφοντας λοιπόν τις προαναφερθέντες τιμές στο txt αρχείο μας με την αντίστοιχη ονομασία KS-VS-Alg0-exp10 που αφορά το πείραμα αυτό, έχουμε ολοκληρώσει την πρώτη, από τις δυο, πειραματική γραμμή.

Για να ολοκληρωθεί μια πειραματική μέτρηση για ένα συγκεκριμένο Ltarget θα πρέπει να συγκρίνουμε τα αποτελέσματα του MAPLE[2] και του MATLAB[1] στις ζητούμενες παραμέτρους (cost, L ,D, κ.λπ..), επομένως, επιστρέφουμε στο command line του Matlab για να εκτελέσουμε τις γνωστές μέχρι τώρα εντολές με μικρές διαφοροποιήσεις που θα εξηγηθούν παρακάτω.

Αρχικά, όπως και σε κάθε πειραματική διαδικασία στο περιβάλλον του MATLAB[1], χρησιμοποιείται η συνάρτηση **stackoutputprint2.** Σε αυτό το πείραμα όμως, χρησιμοποιείται με μια μικρή διαφοροποίηση, το input, δηλαδή την μεταβλητή alpha, το λαμβάνει από το output του

ενθέτου κώδικα alg0. Έχοντας λοιπόν τρέξει την εντολή alg0 έχουμε το εξής αποτέλεσμα:

```
>> alg0
-------estimation reached---------
alpha            Rtarget

ans =

   0.101888656616211   2.820069618225098

LatencyBound          LatencyBound2             minD             maxD

ans =

   5.388668060302735   5.388668060302735   3.732930320570067   7.326639827937418

>>
```

Οπού με κόκκινο χρώμα η μεταβλητή alpha που θα δείξουμε παρακάτω σαν input στη συνάρτηση **stackoutputprint2.**

Επομένως, τρέχουμε την εντολή:

[Vout, tt]= stackoutputprint2(alpha-5/100000, Alpha, costall);

Και οπού alpha εδώ, εννοείται το alpha που έχει τυπώσει το alg0 και διαφέρει από μέτρηση σε μέτρηση.

Θα πρέπει να επισημάνουμε πως δεξιά του input alpha, πρέπει να μειώσουμε κατά περίπου 5/100000 ούτως ώστε η έξοδος tt να γίνει μεγαλύτερη του μηδενός **(tt>0)**.

Παραθέτουμε το συγκεκριμένο στιγμιότυπο οθόνης με σημειωμένα τα alpha που χρησιμοποιούμε για το συγκεκριμένο παράδειγμα

```
>> alg0
-------estimation reached---------
alpha            Rtarget

ans =

   0.101888656616211   2.820069618225098

LatencyBound          LatencyBound2             minD             maxD

ans =

   5.388668060302735   5.388668060302735   3.732930320570067   7.326639827937418

>> knapsack_D2_part1234
>> knapsack_D2_part1234
>> [Vout, tt]= stackoutputprint2(alpha-5/100000, alpha, costall);
```

Έπειτα κατά τα γνωστά πληκτρολογούμε την εντολή **tt** ώστε να πάρουμε output από την συνάρτηση που τρέξαμε και αμέσως μετα το

σύνολο των τριών εντολών που μας δίνουν και το αποτέλεσμα που θέλουμε

A2= Vout(1:tt,:);

A2= sortrows(A2,2);

A2(1,:)

Παρακάτω παρουσιάζονται οι εντολές εκτελεσμένες στο command line του προγράμματος μαζί με τα αποτελέσματα τους

```
>> [Vout, tt]= stackoutputprint2(alpha-5/100000, alpha, costall);
>> tt

tt =

        1365

>> A2= Vout(1:tt,:);
A2= sortrows(A2,2);
A2(1,:)

ans =

   1.0e+02 *

   0.001018470000000   0.124425000000000   0.014517000000000   0.052178100000000   0.038971700000000   3.040000000000000
```

Η τελευταία σειρά είναι αυτή που μας απασχολεί και τη χρησιμοποιούμε σαν μετρό σύγκρισης με τις παραπάνω μεταβλητές που εξάγαμε από το MAPLE[2].

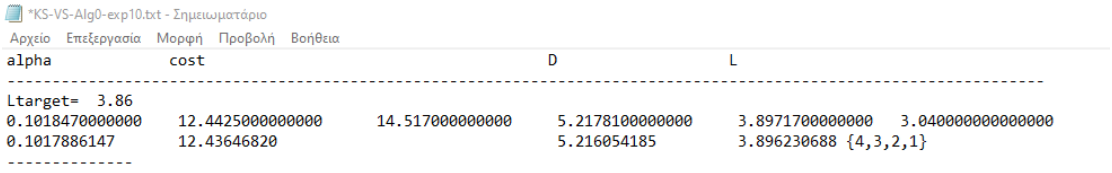
Σημειώνουμε πάλι τις μεταβλητές που αντλούμε και τις επεξηγούμε χρωματικά:

- **a** 1η τιμή (μπλε χρώμα)
- **cost**, 2η τιμή (κόκκινο χρώμα)
- **L** την 5η τιμή (πράσινο χρώμα)
- **D**, 4η τιμή (κίτρινο χρώμα).

```
   1.0e+02 *

   0.001018470000000   0.124425000000000   0.014517000000000   0.052178100000000   0.038971700000000   3.040000000000000

>>
```

Πριν ωστόσο μεταφέρουμε τις παραμέτρους στο τελικό αρχείο txt πρέπει να πολλαπλασιάσουμε τις παραπάνω παραμέτρους με το 100(μετακίνηση υποδιαστολής κατά 2 θέσεις δεξιά), λόγο ενός bug του προγράμματος που παρουσιάζεται σε μερικές τιμές και εμφανίζει τις τιμές διαιρεμένες με το 100, για να έχουμε το ολοκληρωμένο αποτέλεσμα.

Έτσι λοιπόν ολοκληρώνεται μια πειραματική μέτρηση για το 5ο μας πείραμα. Στο τέλος αφού έχουμε μεταφέρει τις τιμές που έχουμε εξάγει έχοντας κάνει το testing στον κώδικα του MAPLE[2] και του MATLAB[1] το αρχείο μας έχει την παρακάτω μορφή:



Οπού αρχικά παρατήσουμε το Ltarget το οποίο μελετάτε, στην περίπτωση μας ο αριθμός 3,86, ακριβώς από κάτω βλέπουμε τα αποτελέσματα του MATLAB[1] και στην τελευταία σειρά αυτά του MAPLE[2].

Κύριος στόχος του πειράματος αυτού είναι η σύγκριση των τιμών στις παραμέτρους που μας απασχολούν, **και κυρίως στο κόστος-cost**, οπού εδώ βλέπουμε πως το **cost MAPLE[2] < cost MATLAB[1]**, που είναι επιθυμητό αποτέλεσμα και ο σκοπός της διαδικασίας. Στις περιπτώσεις που είχαμε **cost MAPLE[2]>cost MATLAB[1]**, σημειώναμε το Ltarget και το αναφέραμε στον υπεύθυνο καθηγητή κ. Αλέξη Καπόρη.

6.4 Συμπεράσματα

Στο κεφάλαιο αυτό μελετήσαμε ένα πείραμα, του οποίου η μεθοδολογία ήταν διαφορετική, δείγμα της πολυπλοκότητας της διπλωματικής εργασίας που φέραμε εις πέρας.

Πρόκειται και πάλι, για ένα πείραμα με δυο γραμμές αποτελεσμάτων, με τα αποτελέσματα να προέρχονται από το MAPLE[2]

και το MATLAB[1], ενώ για πρώτη φορά δεν είχαμε δεδομένα βγαλμένα από τη CLion[3].

Πιο αναλυτικά:

- Η διαδικασία ξεκινάει από το περιβάλλον του MATLAB[1], οπού ο γνωστός πλέον αλγόριθμος **alg0** μας δίνει το **alpha** της 1ης γραμμής, ενώ αμέσως μετα «τρέχουμε», την πολύ σημαντική εντολή, **knapsack_D2_part1234.** Αρμοδιότητα της εντολής αυτής είναι να εντοπίζει τους φακέλους KS1, KS2, KS3 & KS4 και τους "γεμίζει" με δυο φακέλους, τους data & data2 .txt, έπειτα από μερικά δευτερόλεπτα. Τέλος, κατά την επιστροφή στο MATLAB[1] εκτελούμε γνωστές διαδικασίες που έχουν προαναφερθεί.
- Στο περιβάλλον του MAPLE[2] εντοπίζονται οι κυριότερες διάφορες σχετικά με τα προηγούμενα πειράματα. Ανοίγοντας και τα τέσσερα τμήματα κώδικα τα οποία είναι συνδεδεμένα με τα αρχεία στους φακέλους KS1, KS2, KS3 & KS4 ξεκινάμε να τα «τρέχουμε» παράλληλα. Αξίζει να σημειώσουμε πως για να εξάγουν αποτελέσματα τα τέσσερα αυτά αρχεία χρειαζόντουσαν πάρα πολύ χρόνο, κάτι που το επισημάναμε στον υπεύθυνο καθηγητή κ. Καπόρη. Μετα από διορθώσεις το πρόβλημα του χρόνου περιορίστηκε κάπως. Τέλος, στο εύρος τιμών που μελετήθηκε, σε ορισμένες περιπτώσεις, κάποια parts κώδικα έβγαζαν error σαν αποτέλεσμα γεγονός που δεν μας προβλημάτιζε διότι πάντα κάποιο άλλο θα μας έδινε επιθυμητά αποτελέσματα.

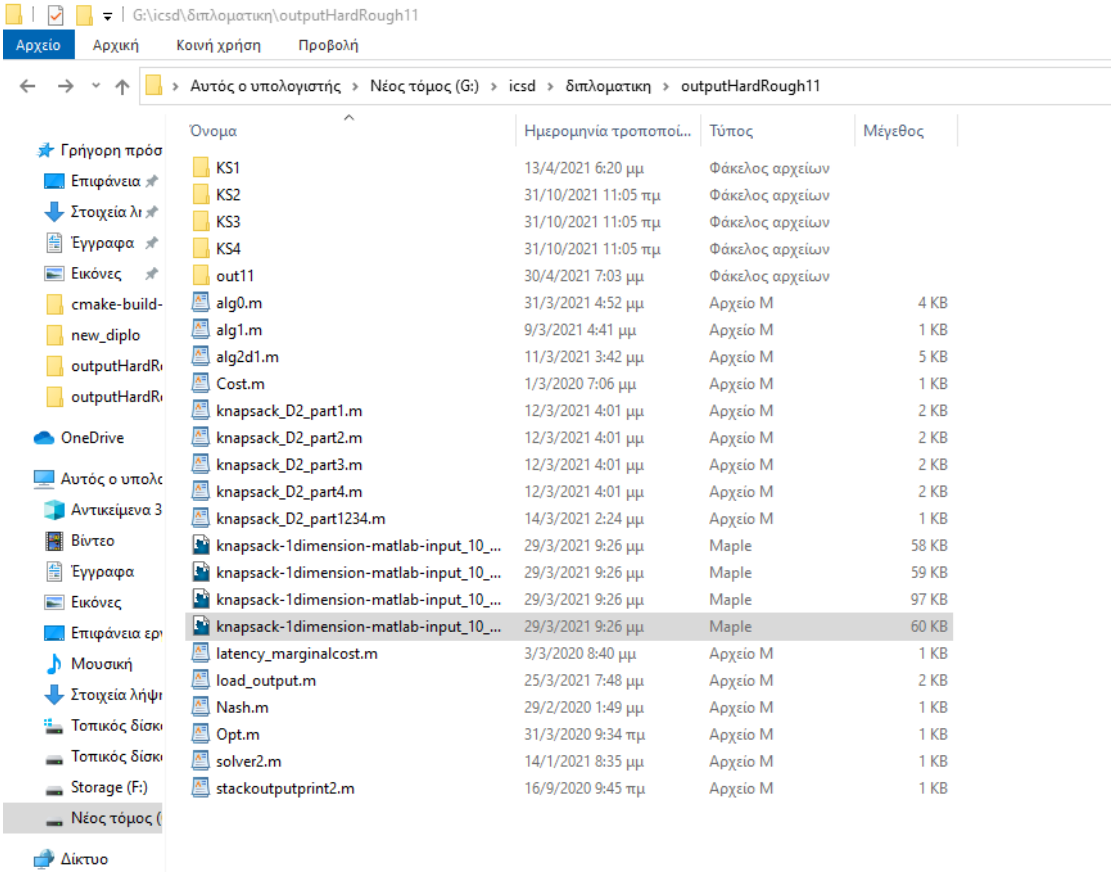
Και στο παρών πείραμα υπήρξαν περιπτώσεις που για το ίδιο **alpha** το **cost** του MAPLE[2] ήταν μεγαλύτερο από αυτό του MATLAB[1]. Τις περιπτώσεις αυτές τις καταγράφαμε ξεχωριστά και τις παραδίδαμε στον υπεύθυνο καθηγητή για περεταίρω διερεύνηση.

Κεφάλαιο 7. Πείραμα 6

7.1 Προεργασία

Μπαίνοντας στο 6ο μας πείραμα, και προ τελευταίο της πειραματικής διαδικασίας, αρκεί να αναφέρουμε πως στο κομμάτι του testing και της όλης εκτέλεσης του έχει αρκετά κοινά σημεία με το προηγούμενο πείραμα 5.

Αρχικά, όπως πάντα, για να ξεκινήσουμε την διαδικασία του testing λαμβάνουμε τα απαραίτητα αρχεία κώδικα από remote του υπευθύνου καθηγητή κ. Αλέξη Καπόρη, καθώς επίσης και τις σχετικές οδηγίες που πρέπει να ακολουθήσουμε ώστε να είναι λειτουργικός αυτός ο κώδικας στους δικούς μας υπολογιστές.



Στο παραπάνω screenshot βλέπουμε το σύνολο των αρχείων που θα πρέπει να μεταφερθούν στους υπολογιστές μας έτσι ώστε να αρχίσουμε τη διαδικασία του testing. Όπως και στο προηγούμενο πείραμα έτσι κι

εδώ ασχολούμαστε με σύγκριση τιμών από τα προγράμματα MAPLE[2] και MATLAB[1].

Από το παραπάνω screenshot αρκεί να πούμε πως τα αρχεία M είναι αρχεία κώδικα με συναρτήσεις που αφορούν τη λειτουργικότητα του πειράματος μας στο MATLAB[1] ενώ τα 4 αρχεία Maple είναι τα 4 parts όπως εξηγήσαμε και στο παραπάνω πείραμα 5, από τα οποία λαμβάνουμε τα αποτελέσματα της 2ης πειραματικής γραμμής σε κάθε Ltarget που μελετάμε. Τέλος, τα αρχεία **KS1 έως KS4** είναι τα φάκελοι που «γεμίζουν» με δεδομένα από το MATLAB[1], όπως θα εξηγήσουμε παρακάτω, και τα οποία δεδομένα χρησιμοποιεί σαν input το εκάστοτε **MAPLE[2] part**

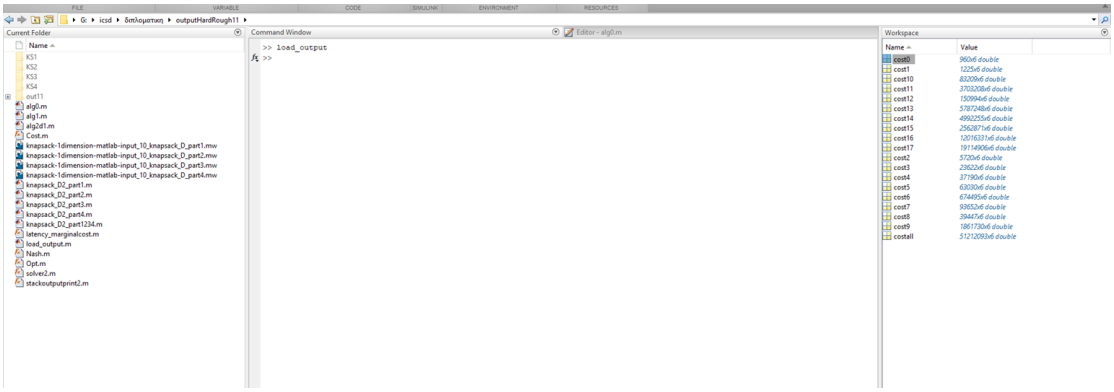
(KS1 για το knapsack part1, KS2 για το knapsack part2… και ούτω καθεξής).

7.2.1 Έναρξη πειραματικής διαδικασίας (MATLAB[1])

Πρώτού ξεκινήσουμε τη διαδικασία, ορίζεται από τον υπεύθυνο καθηγητή το διάστημα στο οποίο θα εξετάζονται τιμές **Ltarget** καθώς επίσης και το βήμα με το οποίο θα προχωράμε από **Ltarget** σε **Ltarget,** στην προκείμενη περίπτωση έχουμε να κάνουμε με τιμές στο διάστημα [2, 3.22] στην μεταβλητή Ltarget, προχωρώντας με μικρά βήματα από την 2 ως την 3.22 με βήμα αύξησης.

Για την γραπτή ανάλυση του *πειράματος 6* χρησιμοποιείται σαν παράδειγμα ο αριθμός **Ltarget=2,26**, τυχαία από το προαναφερόμενο διάστημα τιμών.

Ορίζουμε σαν φάκελο εργασίας στο MATLAB[1] τον νέο μας φάκελο με ονομασία **outputHardRough11,** τα αρχεία του οποίου παρουσιάσαμε παραπάνω, και είμαστε έτοιμοι να εκτελέσουμε την εντολή load_output μια φορά στην αρχή. Θα πρέπει να αναφέρουμε πως είναι μια αρκετά χρονοβόρα διαδικασία λόγο του ότι είναι πολύ μεγάλο το αρχείο με τον κώδικα. Την διαδικασία αυτή την εκτελούμε στην αρχή του πειράματος και έπειτα δεν χρειάζεται να επαναλάβουμε. Το πρόγραμμα μας παίρνει την παρακάτω μορφή όταν είναι όλα έτοιμα για να αρχίσουμε τις μετρήσεις :



Στα αριστερά είναι ο φάκελος εργασίας που προαναφέραμε, στο κέντρο το command line που εισάγουμε εντολές και δεξιά το workspace όπως διαμορφώθηκε με το πέρας της εκτέλεσης **load_output**.

Παράλληλα ανοίγουμε και τον ένθετο κώδικα με ονομασία **alg0,** δεξιά από το command line όπως βλέπουμε παραπάνω, τον οποίο χρησιμοποιούμε για να δώσουμε input τον Ltarget που εξετάζουμε κάθε φορά και σε ψευδοκώδικα είναι της παρακάτω μορφής:

```
INITIALIZE Ltarget to 2.919
INITIALIZE SS to 4
INITIALIZE delta to 1/500
INITIALIZE width to 0
INITIALIZE error1 to 0.000001
INITIALIZE eps to 0.0001
SET format to long

INITIALIZE An to [0.1 0.2 0.15 0.13 0.21 0.1
0.4 0.3]
CALCULATE A_total as the sum of elements in
An
CALCULATE a as [1/An[1], 1/An[2], 1/An[3],
1/An[4], 1/An[5], 1/An[6], 1/An[7], 1/An[8],
3/A_total[1]]
INITIALIZE b as [2 + 0.9, 2 + 0.8, 2 + 0.55, 2 +
0.5, 2 + 0.25, 2 + 0.2, 2 + 0.1, 2 + 0.0, 0]
CALCULATE n as the length of a
CALCULATE r_total as 2 * A_total
CALL alg1
CALL alg2d1
```

Σε κάθε πείραμα που εκτελούμε το πρώτο μας βήμα στο περιβάλλον του MATLAB[1] είναι να ανοίγουμε τον alg0 και διπλά στην μεταβλητή Ltarget να προσθέτουμε τον αριθμό που εκτελείται, στην προκείμενη περίπτωση τον αριθμό 2,26, όπως έχουμε σημειώσει με κόκκινο χρώμα παραπάνω.

Αμέσως μετα εκτελούμε την ομώνυμη εντολή **alg0** στο command line για να πάρουμε τα εξής αποτελέσματα:

```
>> alg0
-------estimation reached---------
alpha            Rtarget

ans =

   0.576131820678711   1.347900810241700

LatencyBound          LatencyBound2              minD              maxD

ans =

                Inf                Inf    3.468960917851772    9.002925417076568

>>
```

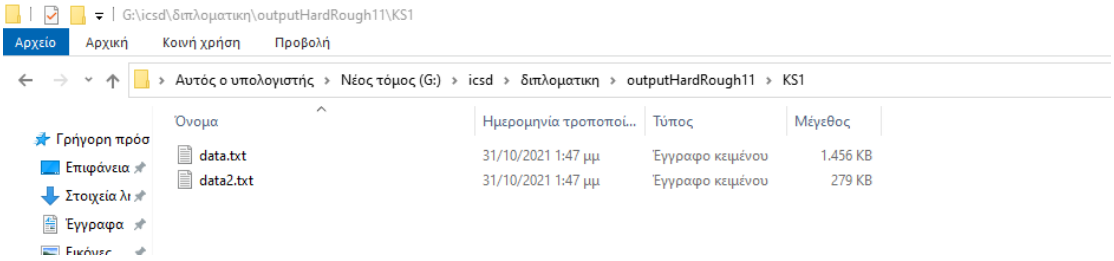
Σημειώσαμε με κόκκινο πλαίσιο τον αριθμό που αποθηκεύεται στη μεταβλητή **alpha** την οποία θα χρησιμοποιήσουμε αργότερα στην συνάρτηση **stackoutputprint2**.

7.2.2 Χρήση τεσσάρων νέων αρχείων κώδικα (MAPLE[2])

Στο σημείο αυτό εκτελούμε την εντολή **knapsack_D2_part1234** η οποία όπως ξέρουμε και από το παραπάνω πείραμα, «γεμίζει» με 2 υποφακέλους, τους φακέλους **KS1-2-3-4.** Απαραίτητη προϋπόθεση να έχουμε ελεεί τους φακέλους αυτούς ώστε να είναι «άδειοι». Ο έλεγχος αυτός είναι απαραίτητος και γίνεται διότι σε περίπτωση που εκτελέσουμε την εντολή **knapsack_D2_part1234** θα προσδεθούν επιπλέον δεδομένα στο φάκελο από 2 διαφορετικά Ltarget και έτσι δε θα πάρουμε σωστά αποτελέσματα από το MAPLE[2].

```
>> knapsack_D2_part1234
>>
```

Εκτελούμε και ξανά ελέγχουμε να δούμε ότι έχουν προστεθεί και τα 2 αρχεία txt και στους **KS1-2-3-4.**



Αφού είναι όλα έτοιμα, ανοίγουμε και τα 4 parts με τον κώδικα του MAPLE[2] και εκτελούμε την εξής διαδικασία:

```
> restart :
  d := 10000 :
  part := 1 :

  data := readdata("./KS1/data.txt", 5) :
  data2 := readdata("./KS1/data2.txt", 8) :

  minCostAll := infinity :
```

Μεταφέρουμε τον κέρσορα στην αρχή του κώδικα διπλά στο restart και πατάμε enter, ενώ από κάτω βλέπουμε το σημείο του κώδικα οπού «διαβάζει» τα δεδομένα που εισήγαμε από το MATLAB[1](το part1 από τον KS1 και ούτω καθεξής.). Σε αυτό το σημείο το πρόγραμμα αρχίζει να κάνει τους μαθηματικούς υπολογισμούς και για το πέρας της διαδικασίας και την εξαγωγή δεδομένων θα πρέπει να ολοκληρώσει 2500 επαναλήψεις.

Πριν συνεχίσουμε με την διαδικασία, θα πρέπει να αναφέρουμε πως η διαδικασία ολοκληρώσεις των επαναλήψεων από το πρόγραμμα Maple για ένα πείραμα κρατούσε παραπάνω από 60 λεπτά και αναφέραμε αυτό το πρόβλημα στον υπεύθυνο καθηγητή κ. Καπόρη με αποτέλεσμα να λάβουμε καινούρια αρχεία MAPLE[2] που βελτίωναν τον χρόνο εκτέλεσης αισθητά. Στα πειράματα που εκτελέστηκαν με τα καινούρια MAPLE[2] αρχεία, αφήναμε ένα σχόλιο *revised MAPLE[2]* για να τα ξεχωρίζουμε, όπως και στο παράδειγμα που εκτελούμε.

Με το που ολοκληρωθούν, λοιπόν, οι επαναλήψεις αυτές, το MAPLE[2] έχει αυτή τη μορφή:

```
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2230, 4.022157778, 5.388668060, 4.366197754, 1.396569072, 12.73933402
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3199, 2231, 4.022157778, 5.388668060, 4.366377439, 1.396861918, 12.73962687
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3199, 2232, 4.022157778, 5.388668060, 4.366557125, 1.397154784, 12.73991973
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3194, 2299, 3.914384232, 5.388668060, 4.478596052, 1.480374294, 12.51921034
                                                                "rep", 2300
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3194, 2300, 3.914384232, 5.388668060, 4.478775737, 1.480786150, 12.51962220
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3195, 2301, 3.914384232, 5.388668060, 4.478955422, 1.481198034, 12.52003408
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3195, 2302, 3.914384232, 5.388668060, 4.479135108, 1.481609944, 12.52044599
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3197, 2303, 3.914384232, 5.388668060, 4.479314794, 1.482021882, 12.52085793
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2304, 3.914384232, 5.388668060, 4.479494479, 1.482433847, 12.52126990
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2305, 3.914384232, 5.388668060, 4.479674165, 1.482845839, 12.52168189
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2306, 3.914384232, 5.388668060, 4.479853850, 1.483257858, 12.52209391
                                                                "rep", 2400
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3194, 2461, 3.990828730, 5.388668060, 4.357705099, 1.435879972, 12.69029482
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3194, 2462, 3.990828730, 5.388668060, 4.357884784, 1.436197114, 12.69061196
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3195, 2463, 3.990828730, 5.388668060, 4.358064469, 1.436514278, 12.69092913
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3196, 2464, 3.990828730, 5.388668060, 4.358244155, 1.436831462, 12.69124631
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3196, 2465, 3.990828730, 5.388668060, 4.358423841, 1.437148667, 12.69156352
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3196, 2466, 3.990828730, 5.388668060, 4.358603526, 1.437465892, 12.69188074
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3197, 2467, 3.990828730, 5.388668060, 4.358783211, 1.437783139, 12.69219799
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2468, 3.990828730, 5.388668060, 4.358962897, 1.438100406, 12.69251526
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2469, 3.990828730, 5.388668060, 4.359142583, 1.438417693, 12.69283254
"SUCCESS, W[rep], W1, rep, Latency_rep, LatencyBound, minLatencyC, cost", 3199, 3198, 2470, 3.990828730, 5.388668060, 4.359322268, 1.438735001, 12.69314985
                                                                "rep", 2500
, Latency[minRep],'min controled L', minLatencyC[minRep], 'D', minD[minRep] + (part - 1) · 2500 · stepsize[minRep] + minRep · (stepsize[minRep]));
```

Έχουν ολοκληρωθεί οι απαραίτητοι υπολογισμοί και αναμένεται από εμάς να μεταφέρουμε τον κέρσορα και να πατήσουμε enter στην εντολή εκτύπωσης ώστε να λάβουμε το σύνολο των μεταβλητών που μας αφορούν. Ελέγχουμε υποχρεωτικά και τα 4 parts και επιλέγουμε να αποθηκεύσουμε το σύνολο των μεταβλητών που μας απασχολούν ανάλογα με το κόστος. Οποίο part έχει το μικρότερο **κόστος-cost** εκείνο επιλέγουμε τελικώς για τους αριθμούς που θα αποθηκεύσουμε.

7.3 Σύγκριση αποτελεσμάτων και συνέχεια της διαδικασίας (επιστροφή στο MATLAB[1])

Σημειώνουμε χρωματικά τους αριθμούς που αντιγράφουμε και αποθηκεύουμε στο αντίστοιχο txt αρχείο που αφορά το συγκεκριμένο πείραμα

Αναλυτικά:

- **a** την τελευταία τιμή (μπλε χρώμα)
- **cost**, όπως προαναφέρθηκε, η 1η τιμή (κόκκινο χρώμα)
- σαν **L** επιλέγουμε την 2η παράμετρο(πράσινο χρώμα)
- σαν **D** επιλέγουμε την τελευταία παράμετρο της 1ης σειράς (κίτρινο χρώμα)
- Τέλος, από τα παραπάνω αποτελέσματα αποθηκεύουμε στο συγκεντρωτικό αρχείο txt αυτού του πειράματος 6 τα **controlled links (μαύρο χρώμα)**

Αντιγράφουμε διαδοχικά τις τιμές στη δεύτερη γραμμή του txt αρχείου μας και ολοκληρώνουμε τη μια εκ των δυο γραμμών.

Την δεύτερη γραμμή την λαμβάνουμε από το MATLAB[1] σύμφωνα με την γνωστή διαδικασία χρησιμοποιώντας για να λάβουμε το απαραίτητο output την συνάρτηση **stackoutputprint2**.

Επιστρέφουμε λοιπόν στο command line του MATLAB[1], και αφού έχουμε υπολογίσει από πριν με τη χρήση της εντολής **alg0**, την μεταβλητή alpha για το Ltarget που μας απασχολεί, εκτελούμε την παρακάτω εντολή:

[Vout, tt]= stackoutputprint2(alpha 5/100000, alpha, costall);

Οπού alpha όπως προαναφέραμε η μεταβλητή από την έξοδο του αλγορίθμου alg0. Ελαττώνουμε τη μεταβλητή alpha ελάχιστα κατά **5/100000** διότι σε αντίθετη περίπτωση η εντολή tt που εκτελούμε αμέσως μετα θα ήταν ίση με το 0.

Αμέσως μετα, εκτελούμε την εντολή tt και θα εμφανιστεί ένας ακέραιος μεγαλύτερος του 0, οπότε επιλέγουμε για ταυτόχρονη εκτέλεση τις τρεις τελευταίες εντολές

A2= Vout(1:tt,:);

A2= sortrows(A2,2);

A2(1,:)

που το MATLAB[1] υπολογίζει και εμφανίζει την ολοκληρωμένη πειραματική γραμμή που πρέπει να κρατήσουμε στο αρχείο μας.

Παραθέτουμε τα βήματα όπως εκτελέστηκαν στο πρόγραμμα:

```
>> [Vout, tt]= stackoutputprint2(alpha-5/100000, alpha, costall);
>> tt

tt =

        4709

>> A2= Vout(1:tt,:);
A2= sortrows(A2,2);
A2(1,:)

ans =

   0.576101000000000   9.766620000000000   6.329470000000000   4.626010000000000   2.549810000000000   5.000000000000000

>>
```

Η τελευταία σειρά είναι αυτή που αφορά το πείραμα μας και από την οποία αντλούμε τις παραμέτρους που θέλουμε.

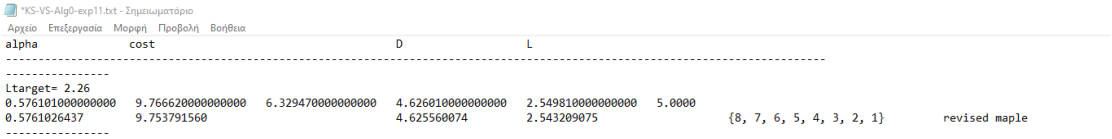
Σημειώνουμε πάλι τις μεταβλητές που αντλούμε και τις επεξηγούμε χρωματικά:

- **a** 1η τιμή (μπλε χρώμα)
- **cost**, 2η τιμή (κόκκινο χρώμα)
- **L** την 5η τιμή (πράσινο χρώμα)
- **D**, 4η τιμή (κίτρινο χρώμα).

```
ans =

   0.576101000000000   9.766620000000000   6.329470000000000   4.626010000000000   2.549810000000000   5.000000000000000

>>
```

Όπως παρατηρούμε στο παράδειγμα αυτό το MATLAB[1] έκανε την εξαγωγή αποτελεσμάτων ορθά, κι έτσι δεν χρειάστηκε να μετακινήσουμε κατά 2 θέσεις την υποδιαστολή στις προαναφερόμενες τιμές

Έτσι λοιπόν, ολοκληρώνεται μια πειραματική μέτρηση για το 6ο στη σειρά πείραμα. Στο τέλος αφού έχουμε μεταφέρει τις τιμές που έχουμε εξάγει έχοντας κάνει το testing στον κώδικα του MAPLE[2] και του MATLAB[1] το αρχείο txt μας έχει την παρακάτω μορφή:



Οπού αρχικά παρατήσουμε το Ltarget το οποίο μελετάμε, στην περίπτωση μας ο αριθμός 2,26, ακριβώς από κάτω βλέπουμε τα αποτελέσματα του MATLAB[1] και στην τελευταία σειρά αυτά του MAPLE[2] μαζί με το αντίστοιχο σχόλιο “revised MAPLE[2]” .

Κύριος στόχος και αυτού του πειράματος αυτού είναι η σύγκριση των τιμών στις παραμέτρους που μας απασχολούν, **και κυρίως στο κόστος-cost**, οπού εδώ βλέπουμε πως το **cost MAPLE[2]<cost MATLAB[1]**, που είναι επιθυμητό αποτέλεσμα και ο σκοπός της διαδικασίας. Στις περιπτώσεις που είχαμε **cost MAPLE[2]>cost MATLAB[1]**, σημειώναμε το Ltarget και το αναφέραμε στον υπεύθυνο καθηγητή κ. Αλέξη Καπόρη.

7.4 Συμπεράσματα

Αφού έχουμε συγκεντρώσει όλα τα δεδομένα στο παρών 6ο πείραμα και έχοντας ολοκληρώσει τη διαδικασία μέσω των MATLAB[1] & MAPLE[2] καταλαβαίνουμε τις ομοιότητες που υπάρχουν με το προηγούμενο πείραμα 5. Επομένως, τα συμπεράσματα τα οποία προκύπτουν είναι ακριβώς τα ιδιά τόσο σε προγραμματιστικό επίπεδο όσο και σε επίπεδο αποτελεσμάτων

Με λίγα λογία:

- Σε κάθε πείραμα ξεκινάμε με την επεξεργασία του αλγορίθμου **alg0** που μας δίνει το **alpha** της 1ης γραμμής, ενώ αμέσως μετα «τρέχουμε», την εντολή, **knapsack_D2_part1234.** Αυτή, όπως προείχαμε, εντοπίζει τους φακέλους KS1, KS2, KS3 & KS4 και τους “γεμίζει” με δυο φακέλους, τους data & data2 .txt, έπειτα από μερικά δευτερόλεπτα. Τέλος, κατά την επιστροφή στο

MATLAB[1] εκτελούμε γνωστές διαδικασίες που έχουν προαναφερθεί.

- Στο MAPLE[2] ανοίγουμε ξανά και τα τέσσερα τμήματα κώδικα τα οποία είναι συνδεδεμένα με τα αρχεία στους φακέλους KS1, KS2, KS3 & KS4. Ξεκινάμε να τα «τρέχουμε» παράλληλα. Και σε αυτό το εύρος τιμών που μελετήθηκε, σε ορισμένες περιπτώσεις, κάποια parts κώδικα έβγαζαν error σαν αποτέλεσμα γεγονός που δεν μας προβλημάτιζε διότι πάντα κάποιο άλλο θα μας έδινε επιθυμητά αποτελέσματα.

Και στο παρών πείραμα υπήρξαν περιπτώσεις που για το ίδιο **alpha** το **cost** του MAPLE[2] ήταν μεγαλύτερο από αυτό του MATLAB[1]. Τις περιπτώσεις αυτές τις καταγράφαμε ξεχωριστά και τις παραδίδαμε στον υπεύθυνο καθηγητή για περεταίρω διερεύνηση.

Κεφάλαιο 8. Πείραμα 7

8.1 Προεργασία

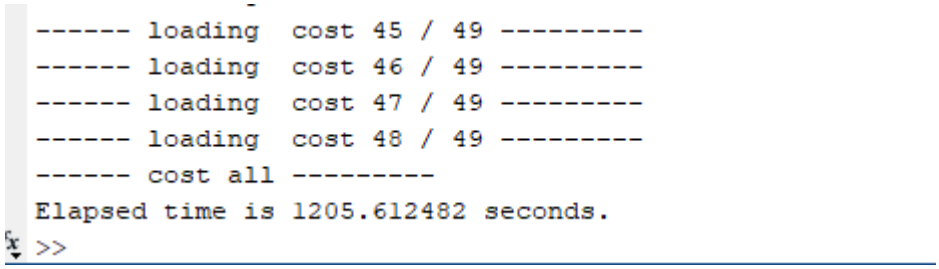
Σχετικά με την προετοιμασία του έβδομου πειράματος, μεταφέραμε μέσω του remote pc,όπου είχαμε πρόσβαση στον Η/Υ του υπεύθυνου καθηγητή κ. Καπόρη ,τον φάκελο **outputHardRough12** που περιέχει το νέο **load_output.m** του MATLAB[1] ,τον φάκελο **out12** από το οποίο αντλεί τα δεδομένα που θα φορτώσει το load_output,το **KS-VS-Alg0-exp12.txt** στο οποίο θα συμπληρώνουμε γραμμή γραμμή τα δεδομένα από τις συγκρίσεις, το αρχείο κώδικα **alg0_CLion[3].m** του MATLAB[1], όπου γεμίζει τον φάκελο στο **cmake-build-release** με 3 αρχεία **txt**, τα οποία θα τα διαβάσει στην συνέχεια το **CLion[3]** ,όπως και τον φάκελο **KSProgram** ο οποίος περιέχει το πρόγραμμα στην Cpp.

Σε αυτό το πείραμα θα χρησιμοποιήσουμε το CLion[3] για την εξαγωγή των δεδομένων της 3ης γραμμής για λόγους ταχύτητας.

Για την λειτουργικότητα του νέου πειράματος, δημιουργούμε νέο φάκελο experiment-12 στον οποίο αντιγράφουμε από το remote PC το νέο **outputHardRough12** και **KSProgram** και οδηγούμε το path του MATLAB[1] στο φάκελο **outputHardRough12**.Επίσης υπάρχει αρχείο κώδικα με όνομα **stackoutputprint2** τον οποίο χρησιμοποιούμε σε όλα τα πειράματα. Αφού ολοκληρωθεί αυτή η διαδικασία τρέχουμε την εντολή **load_output** και περιμένουμε να φορτωθούν τα δεδομένα από τους καταλόγους που υπάρχουν στο φάκελο, όπως ακριβώς και στα προηγούμενα πειράματα. Με το πέρας της διαδικασίας και αφού έχουν γίνει όλα σωστά, το MATLAB[1] είναι έτοιμο να δεχθεί τις κατάλληλες εντολές που θα εξηγηθούν παρακάτω.

```
>> load_output
------loading  cost 0 / 49 ---------
------ loading  cost 1 / 49 ---------
------ loading  cost 2 / 49 ---------
------ loading  cost 3 / 49---------
```

Καταλαβαίνουμε πως έχει ολοκληρωθεί η διαδικασία όταν στο workspace του προγράμματος έχουν φορτωθεί τα ανάλογα κόστη όπως φαίνεται παρακάτω.



8.2.1 Έναρξη πειραματικής διαδικασίας (Εργασία σε MATLAB[1] & CLion[3])

Για την έναρξη της διαδικασίας, ο υπεύθυνος καθηγητής κ. Καπόρης, μας όρισε το διάστημα όπου λαμβάνονται οι μετρήσεις στο 2.25 – 3.75] και με ρυθμό αύξησης 1/1000 . Για να εξηγήσουμε την πειραματική διαδικασία επιλέξαμε τυχαία τον αριθμό Ltarget=2.924 που βρίσκεται μέσα σε αυτό το διάστημα.

Αρχικά μεταφερόμαστε στο περιβάλλον του MATLAB[1] και παράλληλα με το command line του προγράμματος «ανοίγουμε» και το κομμάτι με τον ένθετο κώδικα που έχουμε ονομάσει alg0_CLion[3] και εκεί στο κομμάτι του κώδικα που ονομάζεται Ltarget πάμε και αλλάζουμε κάθε φορά την τιμή και προσθέτουμε οποία εξετάζουμε.

Στην προκείμενη περίπτωση το alg0_CLion[3] σε ψευδοκώδικα είναι της παρακάτω μορφής:

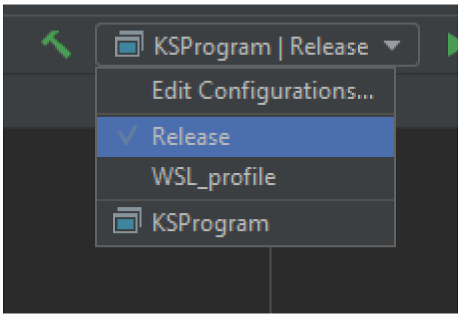
```
INITIALIZE Ltarget to 2.924
INITIALIZE SS to 4
INITIALIZE delta to 10000
INITIALIZE width to 0
INITIALIZE error1 to 0.000001
INITIALIZE eps to 0.0001
SET format to long

INITIALIZE An to [0.6, 0.5, 0.45, 0.15, 0.1, 0.3,
0.5, 0.17, 0.2, 0.23, 0.16, 0.1]
CALCULATE A_total as the sum of elements in
An
CALCULATE a as [1/An[1], 1/An[2], 1/An[3],
1/An[4], 1/An[5], 1/An[6], 1/An[7], 1/An[8],
1/An[9], 1/An[10], 1/An[11], 1/An[12],
3/A_total[1]]
INITIALIZE b as [4.5, 4.3, 4, 3.8, 3.4, 2.9, 2.5,
2, 1.6, 1.1, 0.7, 0.3, 0]
CALCULATE n as the length of a
CALCULATE r_total as 2.2 * A_total
CALL alg1
CALL alg2d1_CLion
CALL knapsack_D_CLion
```

Σε αυτό το σημείο μεταφερόμαστε πίσω στο command line του προγράμματος και πληκτρολογούμε την εντολή alg0 η οποία τρέχει τον ένθετο κώδικα alg0, και άμεσα λαμβάνουμε από το MATLAB[1] τα εξής αποτελέσματα.

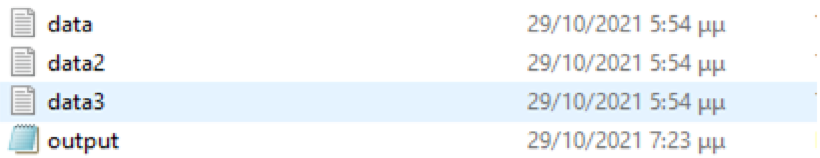
```
>> alg0_CLion
-------estimation reached---------
alpha            Rtarget

ans =

   0.034956932067871   7.345907833099368

LatencyBound          LatencyBound2             minD             maxD

ans =

   4.373005956306849   4.373005956306849   3.084725717304067   4.560108372235094
```

Με το κάλεσμα της alg0_CLion[3] εκτός από τα παραπάνω αποτελέσματα έχουν δημιουργηθεί στον φάκελο KS2,που βρίσκεται μέσα στο cmake-build-release, τρία επιπλέον αρχεία txt τα data data2 data3 τα οποία με την σειρά του θα τα διαβάσει το CLion[3] στο project KSProgram.

Αφού έχουν δημιουργηθεί αυτά τα txt αρχεία, στην συνέχεια ανοίγουμε, τρέχουμε την main.cpp του KSProgram project και επιλέγουμε το “**rebuild all in Release**” έτσι ώστε να προσαρμοστεί το πρόγραμμα στα καινούρια δεδομένα.



Έπειτα τρέχουμε το πρόγραμμα μέσω του CLion[3],το οποίο με την σειρά του μετα το πέρας της ολοκλήρωσης του, θα δημιουργήσει ένα νέο αρχείο .dat με το όνομα output.dat .



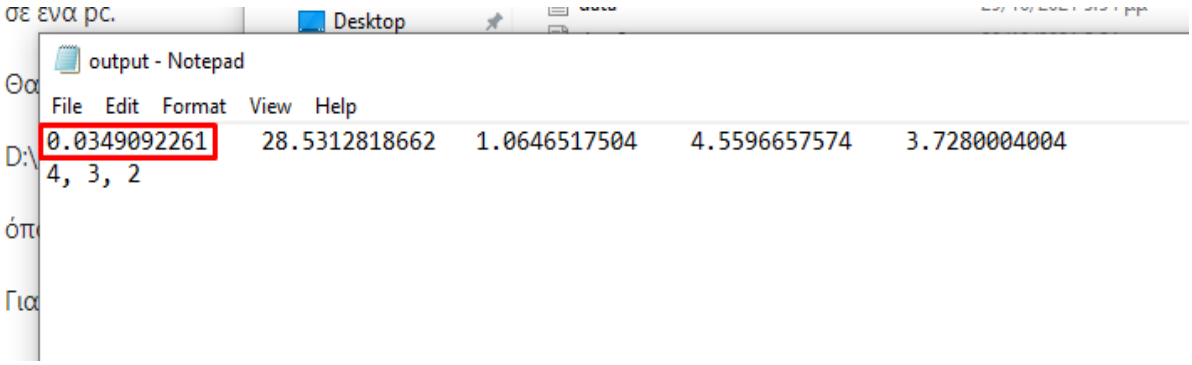
Το συγκεκριμένο αρχείο περιέχει μόνο μία σειρά δεδομένων από την οποία θα δημιουργήσουμε και την 3η γραμμή των αποτελεσμάτων μας, τα οποία αντιγράφουμε στο αρχείο **KS-VS-Alg0-exp12.txt**:

```
-----------
     alpha
Ltarget= 2.924            cost                                  dC                    L
0.3363990000000   27.0478000000000   11.2473000000000   5.5625500000000   3.1279900000000   271.0000000000000
```

Τέλος διαγράφουμε τα περιεχόμενα του φακέλου KS2 διότι πρέπει να είναι κενός για κάθε νέα μέτρηση που θα κάνουμε.

8.2.2 Συμπλήρωση 2ης πειραματικής γραμμής (Εργασία σε MATLAB[1])

Συνέχεια έχει η συμπλήρωση της 2η γραμμής κατά σειρά. Σε αυτή την περίπτωση θα χρησιμοποιήσουμε ως δεδομένο εισαγωγής το alpha από το αρχείο output.dat το οποίο έχουμε ήδη περάσει στο txt με τα αποτελέσματα μας .



Βρισκόμαστε πλέον στο περιβάλλον του MATLAB[1] και αρχικά πληκτρολογούμε την εντολή **format long** για το format των αριθμών. Έπειτα με το πάνω βελάκι του πληκτρολογίου μας αναζητούμε την εντολή **[Vout,tt]=σ(x,x,costall)** , εντολή η οποία καλεί την ομώνυμη συνάρτηση **stackoutputprint2**,όπου ως x βάζουμε το a = 0.03363990429 και πατάμε το enter. Αμέσως μετά πληκτρολογούμε την εντολή **tt** στο command line και παίρνουμε το αποτέλεσμα που αφορά τον αριθμό 0.03363990429. ss από MATLAB[1] το Vout

Τελειώνοντας, επιλέγουμε μαζί τις 3 εντολές

A2= Vout(1:tt,:);

A2= sortrows(A2,2);

A2(1,:);

Στην προκείμενη περίπτωση τα αποτελέσματα μας είναι διαιρεμένα με το 100, από bug του προγράμματος οπότε εμείς μετακινούμε την υποδιαστολή 2 θέσεις μέσα. Τέλος, αποθηκεύουμε τις παραμέτρους **alpha dC** και **L** μαζί με το **cost**, το οποίο αναγράφεται στην δεύτερη τιμή του παραπάνω screenshot.

Αποθηκεύοντας και αυτά τα τελευταία δεδομένα στο συγκεντρωτικό αρχείο μας, έχει ολοκληρωθεί πλήρως μια πειραματική γραμμή και καθ' αυτόν τον τρόπο συνεχίζουμε μέχρις ότου να ολοκληρώσουμε τους αριθμούς μέσα στο διάστημα αυτό.

Το συγκεντρωτικό μας αρχείο είναι αυτής της μορφής:

```
-----------
Ltarget= 2.924
0.3363990000000    27.0478000000000    11.2473000000000     5.5625500000000     3.1279900000000   271.0000000000000
0.3363990429       26.9305186544        9.6854184023        5.4661367924        3.1281332482  8, 7, 6, 5, 4, 3, 2
-----------
```

8.3 Συμπεράσματα

Με το παρών πείραμα ουσιαστικά τελειώνει η πειραματική διαδικασία. Πρόκειται για ένα πείραμα 2 γραμμών αποτελούμενες από δεδομένα τα οποία αντλήθηκαν από το CLion[3] και από το MATLAB[1].

Αναλύοντας τα συμπεράσματα:

- Από το MATLAB[1] με τη χρήση του αλγορίθμου **alg0,** ο οποίος ήταν προσαρμοσμένος για το 7^{ο} πείραμα, αντλούσαμε τα δεδομένα της 1^{ης} πειραματικής γραμμής. Αφού τα αποθηκεύαμε στο γενικό αρχείο txt, επιστρέφαμε στο command line του ίδιου προγράμματος για την ολοκλήρωση και της 2^{ης} γραμμής με τη χρήση της συνάρτησης **stackoutputprint2**. Σκοπός του πειράματος και εδώ ήταν το κόστος της 2^{ης} γραμμής να είναι μικρότερο από το αντίστοιχο της 1^{ης} για παρόμοιο **alpha**.
- Στο κομμάτι της CLion[3], για πρώτη φορά δεν «τρέξαμε» εντολές μέσω Ubuntu[4], αλλά χρησιμοποιήσαμε το ίδιο το πρόγραμμα για να μας εμφανίσει τον φάκελο **output.dat**, από τον οποίο αντλούσαμε το **alpha** για μετέπειτα χρήση.

Κεφάλαιο 9. Χρήσιμες εντολές

9.1 Εισαγωγή

Κατά την διάρκεια της εκτέλεσης της διπλωματικής εργασίας, όπως προαναφέρθηκε, εργαστήκαμε στα εξής προγραμματιστικά περιβάλλοντα:

1. MATLAB[1]
2. CLion[3]
3. MAPLE[2]

Ως εκ τούτου ήρθαμε σε επαφή με νέες και ενδιαφέρουσες εντολές κώδικα αλλά και βιβλιοθήκες που χρησιμοποιήθηκαν για την ολοκλήρωση της πειραματικής διαδικασίας, και στα τρία προγράμματα.

Για το εκάστοτε πρόγραμμα, λοιπόν, θα παρατεθούν οι σημαντικότερες και κυριότερες εντολές που χρησιμοποιήθηκαν με τις επεξηγήσεις τους.

Παραπάνω, εξηγήσαμε την πειραματική διαδικασία βήμα προς βήμα και το πώς χρησιμοποιήθηκε το πρόγραμμα καθώς και τα δεδομένα που αντλήσαμε από αυτό, ενώ ακολουθεί η ανάλυση των

κυριότερων εντολών που χρησιμοποιήθηκαν κατά την εκτέλεση των πειραμάτων, καθώς και τη χρήση αυτών.

9.2 MAPLE[2]

NLPSolve: Χρήση για επίλυση μη γραμμικού προγραγράμματος

OP: Χρήση για εξαγωγή τελεστών από μια έκφραση

Floor: Παράμετρος, η οποία δέχεται έναν πραγματικό αριθμό και επιστρέφει το ακέραιο μέρος του

Add: Χρήση για πρόσθεση συνεχόμενων τιμών

Forced & Free KS: Εμπεριείχαν διανύσματα που χρησιμοποιήθηκαν σαν input σε όλη την πειραματική διαδικασία ανάλογα με τα δεδομένα που προκύπταν από την υπόλοιπη διεργασία για την εξαγωγή δεδομένων.

9.3 MATLAB[1]

Load_output: Η εντολή αυτή φορτώνει τα δεδομένα των υποφακέλων που βρίσκονται στο outputHardRough

stackoutputprint2: Πρόκειται για συνάρτηση που δέχεται ως ορίσματα τα alpha μας ως lowerbound και upperbound**,** όπως και στο **costall** που είναι ένας πίνακας δυσδιάστατος με τα κόστη. Το **vout** είναι ένας αριθμητικός πίνακας που περιέχει τις σειρές από το cost όπου τα στοιχεία στην πρώτη στήλη πέφτουν μεταξύ του lowerbound και upperbound.

Το **T** είναι ένα numeric scalar που αντιπροσωπεύει τον αριθμό των έγκυρων γραμμών στο Vout.

```
function [Vout, t] = stackoutputprint2(lb, ub, cost1)
```

Αρχικοποιείται το Vout ώστε να έχει το ίδιο μέγεθος με το cost1, όπως φαίνεται παρακάτω

```
Vout= zeros(n1,n1b);
t=1;
```

και ύστερα με μία **for** στήλη προς στήλη, εάν το πρώτο στοιχείο αυτής της σειράς του cost1 βρίσκεται εντός ορίων lowerbound και upperbound,δηλαδή του alpha, τότε γεμίζει το Vout με αυτή τη σειρά και αυξάνει τον δείκτη σειράς εξόδου κατά 1.

```
for i=1:nl

if(costl(i,1)>= lb && costl(i,1)<= ub)
Vout(t,:)=costl(i,:);
t=t+1;
end
```

Η παρακάτω σειρά εντολών εκτελούνταν πάντα ταυτόχρονα

A2= Vout(1:tt,:);

A2= sortrows(A2,2);

A2(1,:);

και ουσιαστικά, παίρνει τα δεδομένα από τον πίνακα Vout, τον ταξινομεί σύμφωνα με την δεύτερη στήλη και εμφανίζει τα δεδομένα της πρώτης στήλης.

knapsack_D2_part1234: Καλεί τις knapsack_D2_part1 έως 4 τα οποία παίρνουν σας είδωλο δεδομένα από τα KS1 έως 4 μέσω των φακέλων data και data2.

9.4 Βιβλιοθήκη OpenMP Documentation για το πρόγραμμα Cpp στη CLion[3]

[5] #pragma omp parallel

Σκοπός χρήσης:

Η omp parallel εντολή αναθέτει ρητά τον compiler την παραλληλοποιηση του επιλεγμένου block κώδικα. default (shared | none)

Ορίζει το προεπιλεγμένο πεδίο δεδομένων των μεταβλητών σε κάθε νήμα. Μόνο μία προεπιλεγμένη υποπρόταση μπορεί να καθοριστεί σε μια omp parallel εντολή.

Ο προσδιορισμός της default (none) απαιτεί ότι κάθε μεταβλητή δεδομένων ορατή στο παραλληλισμένο μπλοκ δήλωσης πρέπει να αναφέρεται ρητά σε ένα clause πεδίου δεδομένων, με εξαίρεση τις μεταβλητές που:

-είναι const

-είναι ορισμένες σε μια εμπεριεχόμενη clause χαρακτηριστικών πεδίου δεδομένων.

-χρησιμοποιείται ως μεταβλητή ελέγχου βρόχου που αναφέρεται μόνο από αντίστοιχο omp for ή omp parallel for εντολή.

[6] #pragma omp for

σκοπός χρήσης:

Η omp for εντολή δίνει οδηγία στον μεταγλωττιστή να διανείμει επαναλήψεις βρόχου μέσα στην ομάδα των νημάτων που συναντά αυτό το κατασκεύασμα κοινής χρήσης εργασίας.

collapse (n)

Επιτρέπει την παραλληλοποιηση σε πολλαπλούς βρόχους σε μια εμφώλευση χωρίς να εισαχθεί εμφωλευμένος παραλληλισμό.

-Μόνο ένα collapse επιτρέπεται σε ένα κοινόχρηστο for ή παράλληλο for pragma.

-Ο καθορισμένος αριθμός βρόχων πρέπει να υπάρχει λεξικά. Δηλαδή, κανένας από τους βρόχους δεν μπορεί να είναι σε μια καλούμενη υπορουτίνα.

-Οι βρόχοι πρέπει να σχηματίζουν ένα ορθογώνιο χώρο επανάληψης και τα όρια και το βήμα κάθε βρόχου πρέπει να είναι αμετάβλητα σε όλους τους βρόχους.

-Εάν οι δείκτες βρόχου έχουν διαφορετικό μέγεθος, ο δείκτης με το μεγαλύτερο μέγεθος θα χρησιμοποιηθεί για τον καταρρακτωμένο βρόχο.

-Οι βρόχοι πρέπει να είναι τέλεια εμφωλευμένοι, δηλαδή, δεν υπάρχει παρεμβαλλόμενος κώδικας ούτε κανένα OpenMP pragma μεταξύ των βρόχων που έχουν καταρρεύσει.

-Οι σχετικοί do-βρόχοι πρέπει να είναι δομημένα μπλοκ. Η εκτέλεσή τους δεν πρέπει να τερματίζεται με δήλωση διακοπής.

-Εάν συνδέονται πολλαπλοί βρόχοι με το κατασκεύασμα βρόχου, μόνο μια επανάληψη του εσωτερικού συσχετισμένου βρόχου μπορεί να περιοριστεί με μια δήλωση συνέχισης. Εάν συσχετίζονται πολλαπλοί βρόχοι με το κατασκεύασμα βρόχου, δεν πρέπει να υπάρχουν κλάδοι σε καμία από τις δηλώσεις τερματισμού βρόχου εκτός από τον εσώτατο συσχετισμένο βρόχο.

-Εάν πολλοί βρόχοι συνδέονται με την κατασκευή βρόχου με μια ρήτρα κατάρρευσης, η διατεταγμένη κατασκευή πρέπει να βρίσκεται μέσα σε όλους τους σχετικούς βρόχους.

private (list)

Δηλώνει ότι το εύρος των μεταβλητών δεδομένων στη λίστα είναι ιδιωτικό σε κάθε νήμα. Οι μεταβλητές δεδομένων στη λίστα διαχωρίζονται με κόμματα.

[7] #pragma omp master

σκοπός χρήσης:

Η omp master οδηγία προσδιορίζει μια ενότητα κώδικα που πρέπει να εκτελείται μόνο από το κύριο νήμα.

Χρήση:

Τα νήματα εκτός από το κύριο νήμα δεν θα εκτελέσουν το μπλοκ δήλωσης που σχετίζεται με αυτό το κατασκεύασμα.

Δεν υπάρχει αυτονόητο εμπόδιο είτε στην είσοδο είτε στην έξοδο από το κύριο τμήμα.

[8] #pragma omp barrier

σκοπός χρήσης:

Η οδηγία omp barrier προσδιορίζει ένα σημείο συγχρονισμού στο οποίο τα νήματα σε μια παράλληλη περιοχή δεν θα εκτελεστούν πέρα από το φράγμα omp barrier έως ότου όλα τα άλλα νήματα της ομάδας ολοκληρώσουν όλες τις ρητές εργασίες στην περιοχή.

Χρήση:

Η οδηγία omp barrier πρέπει να εμφανίζεται μέσα σε ένα μπλοκ ή δηλωμένη ένωση.

[9] #pragma omp critical

σκοπός χρήσης:

Η εντολή omp critical προσδιορίζει ένα τμήμα κώδικα που πρέπει να εκτελείται από ένα μόνο νήμα κάθε φορά.

Χρήση:

Ένα νήμα περιμένει στην αρχή μιας κρίσιμης περιοχής που προσδιορίζεται από ένα δοσμένο όνομα έως ότου κανένα άλλο νήμα στο πρόγραμμα δεν εκτελεί μια κρίσιμη περιοχή με το ίδιο όνομα. Οι κρίσιμες ενότητες που δεν κατονομάζονται συγκεκριμένα από την omp critical αντιστοιχίζονται στο ίδιο απροσδιόριστο όνομα.

Παραδείγματα χρήσης των συναρτήσεων OMP

Υπολογίζει το άθροισμα 1+2+3….+9=45 με παραλληλοποίηση

```
// shared variable
int sum_shared = 0;
#pragma omp parallel
{
    // private variables (one for each thread)
    int sum_local = 0;
    #pragma omp for nowait
    for (int i = 0; i < 10; ++i) {
        sum_local += i;
```

Υπολογίζει το άθροισμα ενός πίνακα με παραλληλοποίηση

```
1.  #include <iostream>
2.  #include <omp.h>
3.  using namespace std;
4.
5.  int summation(int *arr,int n){
6.  if(n==0)
7.  return 0;
8.  if(n==1)
9.  return *arr;
10. else
11. return summation(arr,(n/2))+summation(arr+(n/2),n-(n/2));
12. }
13.
14. int main()
15. {
16. int sum=0,sum1=0,sum2=0,size=100;
17. int arr[size];
18. for(int i=0;i<size;i++)
19. arr[i]=i+1;
20.
21. #pragma omp parallel
22. #pragma omp single
23. {
24.     #pragma omp task shared(sum1)
25.     sum1=summation(arr,size/2);
26.     #pragma omp task shared(sum2)
27.     sum2=summation(arr+size/2,size/2);
28.
29.     #pragma omp taskwait
30.     sum=sum1+sum2;
31. }
32. cout<<sum<<endl;
33.
34. return 0;
35. }
36.
```

Βιβλιογραφία

[1]Matlab for Windows,1994-2022 The MathWorks,Inc.
https://www.mathworks.com/products/matlab.html

[2]Maplesoft, a division of Waterloo Maple Inc. 2022
https://www.maplesoft.com/products/Maple/students/

[3]2000-2022 JetBrains s.r.o
https://www.jetbrains.com/clion/

[4] 2022 Canonical Ltd. Ubuntu and Canonical,registered trademarks of Canonical Ltd.

https://ubuntu.com/wsl

OpenMP Functions

[5] IBM, pragma omp parallel

[6] IBM, oragma omp for

[7] IBM, pragma omp master

[8] IBM, pragma omp barrier

[9] IBM, pragma omp critical

MATLAB Functions

[10] length(obj)

[11] options=optimset(Name,value)

[12] M=min(A)

[13] S=sum(A)

[14] b=mod(a,m)

[15] Y=ceil(X)

[16] X=zeros(n)

[17] W=width(T)

[18] X=ones(n)

[19] d=eps(x)

[20] x=fmincin(fun,x0,A,B)

Maple Functions

[21] NLPSolve(obj, constr, db, opts)

[22] op(i, e)

[23] floor(M)

[24] min(x1, x2, …)

[25] max(x1, x2, …)

[26] seq(m .. n, step)

[27] Add(A, B, c1, c2, ip, options)

[28] readdata(fileID, n)

[29] fsolve(equations, variables, complex)

Maple Packages

[30] Optimization package

[31] MTM package

Other References

[32] Stackelberg Competition

[33] The Price of Optimum in Stackelberg Games

[34] Experimental Results for Stackelberg Scheduling Strategies

[35] Game Theory Through Examples

[36] Solution Manual Game Theory: An Introduction