ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΩΝ ΓΙΑ ΤΟΝ ΥΠΟΛΟΓΙΣΜΟ ΤΕΤΡΑΓΩΝΙΚΗΣ ΡΙΖΑΣ ΜΕ ΤΗΝ ΒΟΗΘΕΙΑ ΤΗΣ **JULIA**

NEWTON-RAPHSON vs ΠΡΟΣΕΓΓΙΣΗ ΜΕ ΠΑΡΑΛΛΗΛΟΓΡΑΜΜΑ

Υπεύθυνη καθηγήτρια: κ.Μητρούλη
Επιμέλεια Εργασίας:Θεοχάρης-Κρεμμύδας Μάρκος, Βαλίνου Χρυσούλα

Contents

1 ΕΙΣΑΓΩΓΗ

Στο μάθημα της Πληροφορικής Ι, το οποίο διδαχθήκαμε στο 1ο εξάμηνο φοίτησής μας στο Τμήμα Μαθηματικών ΕΚΠΑ και στην ενότητα συναρτήσεις, ένα βασικό παράδειγμα μελέτης που μας απασχόλησε αρκετά ήταν ο υπολογισμός της τετραγωνικής ρίζας με διάφορες μεθόδους, με σκοπό την δημιουργία του κατάλληλου αλγορίθμου για τον υπόλογισμό της. Οι μεθόδοι που αναπτύξαμε για τον υπολογισμό της τετραγωνικής ρίζας ήταν οι εξής:
α) Μέθοδος Newton-Raphson
β) Γεωμετρική προσέγγιση με την μέθοδο των παραλληλογράμμων .
Τότε γνωρίζαμε την γλώσσα προγραμματισμού Octave, όμως στην συνέχεια ύστερα από έρευνα μας για πιο γρήγορες και σύγχρονες γλώσσες προγραμματισμού και με την βοήθεια της υπεύθυνης καθηγήτριας μας κ. Μητρούλη γνωρίσαμε και αξιοποιήσαμε την γλώσσα προγραμματισμού Julia, μια γλώσσα δυναμικού προγραμματισμού υψηλού επιπέδου και υψηλής απόδοσης και υλοποιήσαμε τους υπολογισμούς μας σε αυτή.

Στη συνέχεια παρουσιάζουμε τη σύγκριση των δύο αλγορίθμων για τον υπολογισμό της τετραγωνικής ρίζας ενός αριθμού. Ο πρώτος είναι η μέθοδος Newton-Raphson, με μεταβλητές εισόδου τον αριθμό x του οποίου ζητείται προσέγγιση της ρίζας, μια αρχική προσέγγιση της τετραγωνικής ρίζας του x, καθώς και το μέγιστο επιτρεπόμενο σφάλμα ή το πλήθος των επαναλήψεων (iterations), ανάλογα με την παράμετρο που θέλουμε να εξετάσουμε. Η δεύτερη προσεγγίζει τη ρίζα του x μέσω ορθογώνιων παραλληλόγραμμων, εμβαδού x. Συγκεκριμένα, καθώς τα μήκη των πλευρών του παραλληλόγραμμου τείνουν να εξισωθούν (και το παραλληλόγραμμο τείνει να γίνει τετράγωνο), το μήκος της πλευράς προσεγγίζει την sqrt(x). Σαν μεταβλητές εισόδου, δέχεται το x και το μέγιστο επιτρεπόμενο σφάλμα ή το πλήθος των επαναλήψεων.

Η σύγκριση των αλγορίθμων πραγματοποιήθηκε ως προς δύο παραμέτρους:
a) Ως προς το σχετικό σφάλμα ύστερα από ορισμένο αριθμό επαναλήψεων.
b) Ως προς τον απαιτούμενο χρόνο για τη συμπλήρωση ορισμένου πλήθους επαναλήψεων.

2 Μέθοδος Newton-Raphson

2.1 Ιστορικά στοιχεία

Το όνομα "μέθοδος του Νεύτωνα" προέρχεται από την περιγραφή του Ισαάκ Νεύτωνα μιας ειδικής περίπτωσης της μεθόδου στο De analysi per aequationes numero terminorum infinitas (γραμμένο το 1669, που δημοσιεύτηκε το 1711 από τον William Jones) και στο De metodis fluxionum et serierum infinitarum (γράφτηκε το 1671, μεταφράστηκε και δημοσιεύτηκε ως Method of Fluxions το 1736 από τον John Colson). Ωστόσο, η μέθοδός του διαφέρει ουσιαστικά από τη σύγχρονη μέθοδο που περιγράφουμε στη συνέχεια. Ο Newton εφάρμοσε τη μέθοδο μόνο σε πολυώνυμα, ξεκινώντας με μια αρχική εκτίμηση ρίζας και εξάγοντας μια ακολουθία διορθώσεων σφαλμάτων. Χρησιμοποίησε κάθε διόρθωση για να ξαναγράψει το πολυώνυμο ως προς το υπόλοιπο σφάλμα, και στη συνέχεια υπολόγισε μια νέα διόρθωση παραβλέποντας όρους υψηλότερου βαθμού. Δεν συνέδεσε ρητά τη μέθοδο με παράγωγους ούτε παρουσίασε έναν γενικό τύπο. Ο Newton εφάρμοσε αυτή τη μέθοδο τόσο σε αριθμητικά όσο και σε αλγεβρικά προβλήματα, παράγοντας σειρές Taylor στην τελευταία περίπτωση.

Ο Newton μπορεί να άντλησε τη μέθοδό του από μια παρόμοια αλλά λιγότερο ακριβή μέθοδο του Vieta. Η ουσία της μεθόδου του Vieta μπορεί να βρεθεί στο έργο του Περσή μαθηματικού Sharaf al-Din al-Tusi, ενώ ο διάδοχός του Jamshid al-Kashi χρησιμοποίησε μια μορφή της μεθόδου του Νεύτωνα για να λύσει την $x^P - N = 0$ και να βρει τις ρίζες ως προς N. Μια ειδική περίπτωση της μεθόδου του Νεύτωνα για τον υπολογισμό των τετραγωνικών ριζών ήταν γνωστή από την αρχαιότητα και συχνά ονομάζεται Βαβυλωνιακή μέθοδος.

Η μέθοδος του Νεύτωνα χρησιμοποιήθηκε από τον Ιάπωνα μαθηματικό του 17ου αιώνα Seki Kōwa για να λύσει εξισώσεις μίας μεταβλητής, αν και η σύνδεση με τον απειροστικό λογισμό έλειπε.

Η μέθοδος του Νεύτωνα δημοσιεύτηκε για πρώτη φορά το 1685 στο A Treatise of Algebra both Historical and Practical από τον John Wallis. Το 1690, ο Joseph Raphson δημοσίευσε μια απλοποιημένη περιγραφή στο Analysis aequationum universalis. Ο Raphson εφάρμοσε επίσης τη μέθοδο μόνο σε πολυώνυμα, αλλά απέφυγε την κουραστική διαδικασία επανεγγραφής του Newton εξάγοντας κάθε διαδοχική διόρθωση από το αρχικό πολυώνυμο. Αυτό του επέτρεψε να παράγει μια επαναχρησιμοποιήσιμη επαναληπτική έκφραση για κάθε πρόβλημα. Τέλος, το 1740, ο Thomas Simpson περιέγραψε τη μέθοδο του Νεύτωνα ως μια επαναληπτική μέθοδο για την επίλυση γενικών μη γραμμικών εξισώσεων χρησιμοποιώντας απειροστικό λογισμό, δίνοντας ουσιαστικά την παραπάνω περιγραφή. Στην ίδια δημοσίευση, ο Simpson δίνει επίσης τη γενίκευση σε συστήματα δύο εξισώσεων και σημειώνει ότι η μέθοδος του Newton μπορεί να χρησιμοποιηθεί για την επίλυση προβλημάτων βελτιστοποίησης θέτοντας την κλίση ίση με μηδέν.

Ο Arthur Cayley το 1879 στο The Newton-Fourier imaginary problem ήταν ο πρώτος που παρατήρησε τις δυσκολίες στη γενίκευση της μεθόδου του Newton σε μιγαδικές ρίζες πολυωνύμων με βαθμό μεγαλύτερο από 2 και μιγαδικές αρχικές τιμές. Αυτό άνοιξε το δρόμο για τη μελέτη της θεωρίας των επαναλήψεων των ρητών συναρτήσεων.

2.2 Περιγραφή μεθόδου

Η μέθοδος των Newton-Raphson είναι η πιο "δημοφιλής" μέθοδος για τη λύση μη γραμμικών εξισώσεων, διότι είναι απλή στην εφαρμογή της και επί πλέον για απλές ρίζες είναι τετραγωνικής σύγκλισης. Η μέθοδος Newton-Raphson είναι μια ειδική περίπτωση της γενικής επαναληπτικής μεθόδου.

Έστω η εξίσωση $f(x) = 0$ **(1)**, όπου f γνωστή συνάρτηση συνεχής με πραγματικούς συντελεστές, γνήσια μονότονη και x η ανεξάρτητη μεταβλητή. Θέλουμε να βρούμε τη ρίζα ξ της εξίσωσης σε ένα διάστημα όπου η ρίζα είναι απλή. Από το ανάπτυγμα Taylor έχουμε:

$f(\xi) = f(x_n) + (\xi - x_n)nf'(x_n) + \frac{1}{2}(\xi - x_n)f"(h)$ **(2)**,όπου

h $\in (\min \xi, x_n, \max \xi, x_1)$

Αφού το ξ είναι ρίζα της **(1)**, τότε $f(\xi) = 0$, επίσης υποθέτουμε ότι $\lim_{n\to\infty} x_{n+1} = \xi$ και κρατώντας από την **(2)** τους δύο πρώτους όρους του αναπτύγματος έχουμε:

$$0 = f(x_n) + (x_{n+1} - x_n)f'(x_n) \ \textbf{(3)} \Leftrightarrow$$

$$\Leftrightarrow x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \textbf{(4)} \text{ ,για } n = 1, 2, ...$$

Κατασκευάστηκε ένας προσεγγιστικός επαναληπτικός αλγόριθμος που αποτελεί τον αλγόριθμο των Newton-Raphson.

2.3 Χρήση του αλγορίθμου Newton-Raphson για την εύρεση ριζών πραγματικού αριθμού

Έστω η εξίσωση $x^v = A$ **(5)**, όπου Α πραγματικός αριθμός και ν φυσικός αριθμός και θέλουμε να υπολογίσουμε τη ρίζα της εξίσωσης αυτής, δηλαδή την $\sqrt[v]{A}$ ν-ιοστή ρίζα του Α.
Παίρνουμε την εξίσωση: $f(x) = x^v - A = 0$ **(6)**

Υπολογίζουμε την: $f'(x) = \nu * x^{\nu-1}$ **(7)**

Αντικαθιστούμε τις σχέσεις **(6)**,**(7)** στον τύπο **(4)** του αλγορίθμου Newton – Raphson και έχουμε:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Ο αλγόριθμος αυτός αποτελεί τον αλγόριθμο των Newton – Raphson για την εύρεση της ν-ιοστής ρίζας του Α$\in \mathbb{R}$, με χ_0 κατάλληλη αρχική προσέγγιση. Ειδικά για ν=2, έχουμε να υπολογίσουμε τη ρίζα της εξίσωσης $x^2 = A$, δηλαδή την $\sqrt[2]{A}$ τετραγωνική ρίζα του Α και για την $f(x) = x^2 - A = 0$,ο αλγόριθμος των Newton-Raphson παίρνει τη μορφή:

$x_{n+1} = \frac{1}{2}(x_n + \frac{A}{x_n})$ **(8)** , για $n = 1, 2, ...$ όπου x_0 κατάλληλη αρχική προσέγγιση.

2.4 Γεωμετρική Ερμηνεία της Μεθόδου Newton-Raphson

Θεωρούμε τη συνάρτηση $y = f(x)$ της οποίας η γραφική παράσταση φαίνεται στο σχήμα 1. Το σημείο τομής της καμπύλης $y = f(x)$ με τον άξονα των x είναι προφανώς η ρίζα της εξίσωσης $f(x) = 0$.
Έστω ένα σημείο x_κ. Η κάθετος στον άξονα των x στο σημείο x_κ k τέμνει την καμπύλη, στο σημείο Α. Στο Α φέρνουμε την εφαπτομένη στην καμπύλη η οποία τέμνει τον άξονα των x στο $x_{\kappa+1}$. Αν συνεχίσουμε την ίδια διαδικασία για το $x_{\kappa+1}$ βρίσκουμε το σημείο $x_{\kappa+2}$ κ.ο.κ. μέχρι που να προσεγγίσουμε το σημείο α με την επιθυμητή ακρίβεια.

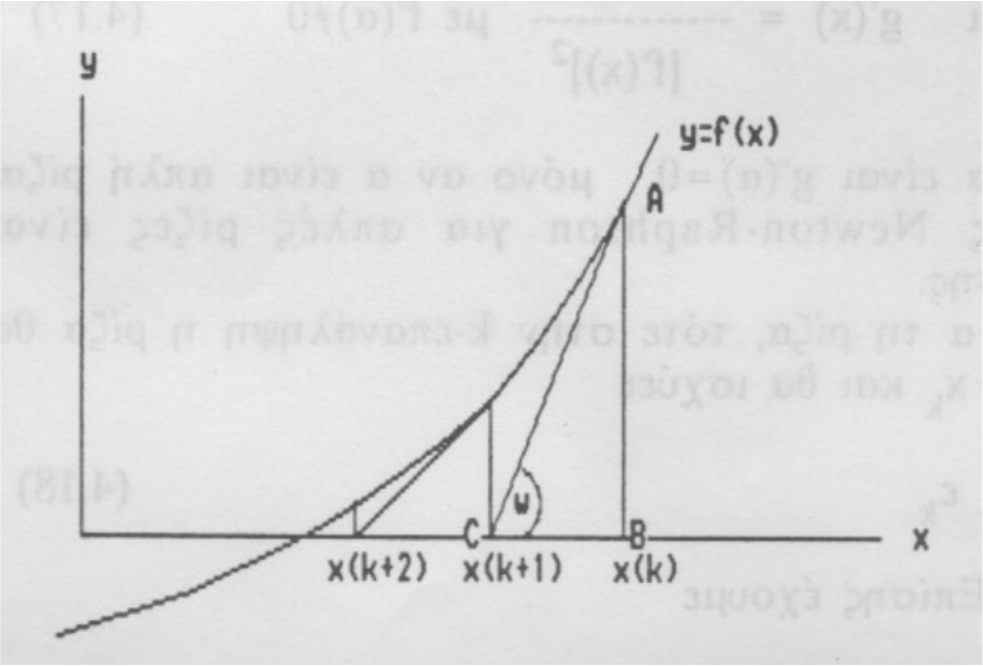


Figure 1

Από το τρίγωνο ABC έχουμε τη σχέση: $\tan(\omega) = \dfrac{(AB)}{(BC)}$.

Επειδή $(AB) = f(x_\kappa), (BC) = x_\kappa - x_{\kappa+1}$ εύκολα καταλήγουμε στον τύπο των Newton-Raphson

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Δηλαδή με τη μέθοδο Newton-Raphson προσεγγίζουμε την καμπύλη σε κάθε σημείο με την εφαπτομένη στο σημείο αυτό.
Στη προσεγγιστική αριθμητική του υπολογιστή θεωρούμε ότι βρήκαμε την ρίζα

όταν φράξουμε το σχετικό σφάλμα από κάποιο πολύ μικρό αριθμό ε (επιθυμητή ακρίβεια). Δηλ. όταν $|x^2 - a|/x^2 = |x - a/x|/x < \varepsilon$.

3 Γεωμετρική προσέγγιση με την μέθοδο των παραλληλογράμμων

3.1 Ιστορικά στοιχεία

Ίσως ο πρώτος αλγόριθμος που χρησιμοποιήθηκε για την προσέγγιση $\sqrt{S}$ είναι γνωστή ως Βαβυλωνιακή μέθοδος, παρόλο που δεν υπάρχουν άμεσες αποδείξεις πέρα από τεκεμεριωμένη εικασία ότι οι επώνυμοι Βαβυλώνιοι μαθηματικοί χρησιμοποίησαν αυτή τη μέθοδο. Η μέθοδος είναι επίσης γνωστή ως μέθοδος του Ήρωνα, από τον Έλληνα μαθηματικό του πρώτου αιώνα Ήρωνα της Αλεξάνδρειας που έδωσε την πρώτη ρητή περιγραφή της μεθόδου στο έργο του Metrica του 60 μ.Χ. Η βασική ιδέα είναι ότι αν το x είναι υπερεκτίμηση της τετραγωνικής ρίζας ενός μη αρνητικού πραγματικού αριθμού S τότε $\frac{S}{x}$ θα είναι μια υποεκτίμηση, ή το αντίστροφο, και έτσι ο μέσος όρος αυτών των δύο αριθμών μπορεί εύλογα να αναμένεται να παρέχει μια καλύτερη προσέγγιση (αν και η επίσημη απόδειξη αυτού του ισχυρισμού εξαρτάται από την ανισότητα των αριθμητικών και γεωμετρικών μέσων που δείχνει ότι αυτός ο μέσος όρος είναι πάντα υπερεκτίμηση της τετραγωνικής ρίζας, όπως σημειώνεται στο άρθρο για τις τετραγωνικές ρίζες, διασφαλίζοντας έτσι τη σύγκλιση). Αυτό ισοδυναμεί με τη χρήση της μεθόδου του Νεύτωνα για επίλυση της $x^2 - S = 0$.

Πιο συγκεκριμένα, αν x είναι η αρχική μας εικασία για την τιμή του $\sqrt{S}$ και ε είναι το σφάλμα στην εκτίμησή μας έτσι ώστε $S = (x + \varepsilon)^2$, τότε μπορούμε να επεκτείνουμε το διώνυμο και να λύσουμε: $\varepsilon = \frac{S - x^2}{2x + \varepsilon} \approx \frac{S - x^2}{2x}$, από ε<<x.

3.2 Περιγραφή μεθόδου

Ο υπολογισμός της τετραγωνικής ρίζας ενός θετικού αριθμού x ισοδυναμεί με την "κατασκευή" ενός τετραγώνου εμβαδόν x. Προσεγγίζοντας γεωμετρικά το πρόβλημα, κάνουμε τις εξής παρατηρήσεις. Πρώτον, αν ένα τετράγωνο με πλευρά Α έχει το ίδιο εμβαδόν με ένα Κ× Β ορθογώνιο, τότε όπως φαίνεται και στο παρακάτω σχήμα 2, η τετραγωνική ρίζα του Α είναι μεταξύ των Κ και Β. Δεύτερον, μπορούμε να κάνουμε ένα δεδομένο ορθογώνιο "πιο τετράγωνο" αντικαθιστώντας το Β=$\frac{(B+K)}{2}$ και Κ=$\frac{K}{2}$. Η διαδικασία μπορεί προφανώς να επαναληφθεί. Η διαδικασία υπολογισμού σταματά όταν φθάσουμε σε κάποιο ορθογώνιο που μοιάζει αρκετά με τετράγωνο.

3.3 Γεωμετρική Ερμηνεία της Μεθόδου των Παραλληλογράμμων

Εδώ απεικονίζεται σχηματικά η παραπάνω διαδικασία.

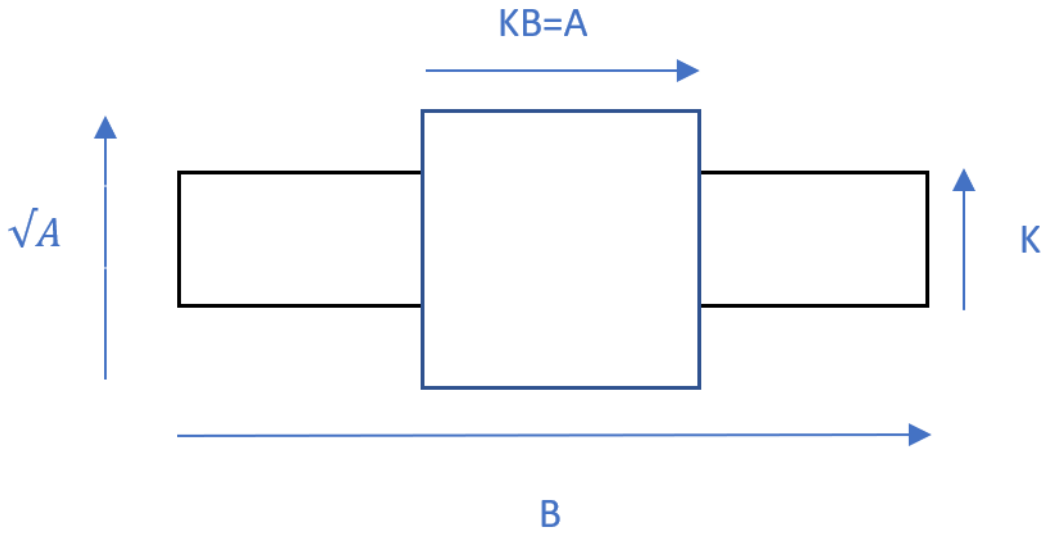


Figure 2

4 Υλοποίηση σε Julia

Οι δύο αλγόριθμοι που παρουσιάσαμε μπορούν να υλοποιηθούν σε περιβάλλον Julia με τον ακόλουθο τρόπο:

```
function rect(a, N)
L = a
W = 1
for i=1:N
W = (L+W)/2
L = a/W
end
return W
end
```

```
function NewtonRaphson(a, N, x)
sqa = x
for i=1:N
sqa = (sqa + a/sqa)/2
end
return sqa
end
```

Παρατηρούμε ότι η μεταβλητή εισόδου a είναι ο αριθμός του οποίου προσεγγίζουμε τη ρίζα, η μεταβλητή εισόδου N είναι ο αριθμός των επαναλήψεων που πραγματοποιούνται κατά την εκτέλεση των μεθόδων και η μεταβλητή εισόδου x είναι η αρχική εκτίμηση της ρίζας που παρέχεται στη μέθοδο Newton-Raphson. Θα υιοθετήσουμε τη χρήση αυτών των τριών συμβόλων για το υπόλοιπο του εγγράφου.

5 Σύγκριση των δύο μεθόδων ως προς το σχετικό σφάλμα

Προκειμένου να συγκρίνουμε τη μέθοδο Newton-Raphson με τη μέθοδο των ορθογωνίων, θα πρέπει να λάβουμε υπόψη διάφορες παραμέτρους που ενδέχεται να επηρεάζουν το σχετικό τους σφάλμα. Για παράδειγμα, αναμένουμε το ποια μέθοδος έχει το μικρότερο σχετικό σφάλμα ύστερα από ορισμένο αριθμό επαναλήψεων να εξαρτάται από την αρχική προσέγγιση της ρίζας που παρέχεται στον αλγόριθμο Newton-Raphson. Επιπλεόν, για σταθεροποιημένη τιμή της αρχικής προσέγγισης, είναι πιθανό για ορισμένες τιμές του αριθμού a του οποίου προσεγγίζεται η ρίζα να έχει (ύστερα από ορισμένο αριθμό επαναλήψεων) μικρότερο σφάλμα η μέθοδος Newton-Raphson, ενώ για άλλες να έχει μικρότερο σφάλμα η μέθοδος των ορθογωνίων. Τέλος, μια ακόμα παράμετρος που ενδέχεται να επηρεάζει τη σύγκριση των δύο μεθόδων είναι ο αριθμός των επαναλήψεων N: θα μπορούσε, για παράδειγμα, αρχικά (δηλ. για μικρές τιμές του N) να έχει μικρότερο σφάλμα η μία μέθοδος, ενώ ύστερα από έναν αριθμό επαναλήψεων να γίνεται μικρότερο το σχετικό σφάλμα της άλλης. Σκοπός αυτής της ενότητας είναι η διερεύνηση του τρόπου με τον οποίο επηρεάζουν αυτές οι παράμετροι τη σύγκριση των δύο μεθόδων ως προς το σχετικό σφάλμα.

Θα ξεκινήσουμε με την τελευταία εξ αυτών, δηλαδή τον αριθμό των επαναλήψεων (N). Συγκεκριμένα, θα σταθεροποιήσουμε προσωρινά τα a και x, οπότε θα συγκρίνουμε τα σχετικά σφάλματα των δύο μεθόδων για διάφορες τιμές του N. Ύστερα, θα επαναλάβουμε τη διαδικασία και για άλλες τιμές των a και x και θα καταλήξουμε σε συμπεράσματα.

Για την παραγωγή των δεδομένων, δημιουργούμε δύο συναρτήσεις που επιστρέφουν το σχετικό σφάλμα ύστερα από N επαναλήψεις (μία συνάρτηση για τη μέθοδο Newton-Raphson και μία για τη μέθοδο των ορθογωνίων), για συγκεκριμένες τιμές των παραμέτρων N, a και x:

```
function err_rect(a, N)
approx = rect(a, N)
sqa = sqrt(a)
return abs(approx-sqa)/sqa
end
```

```
function err_NR(a, N, x)
approx = NewtonRaphson(a, N, x)
sqa = sqrt(a)
return abs(approx-sqa)/sqa
end
```

Στη συνέχεια ορίζουμε άλλη μία συνάρτηση, η οποία, καλώντας μέσα σε ένα for loop τις παραπάνω για διαφορετικές τιμές των παραμέτρων τους, δημιουργεί τα δεδομένα και τα εκτυπώνει στην επιθυμητή μορφή (εδώ σε μορφή πίνακα, του οποίου τα στοιχεία διαχωρίζονται με κόμμα):

```
function err_comp(a, N_range, x_range)
print(", ")
for N in N_range
print("N=", N, ", ")
end
print('\n', "Rect, ")
for N in N_range
print(err_rect(a, N), ", ")
end
```

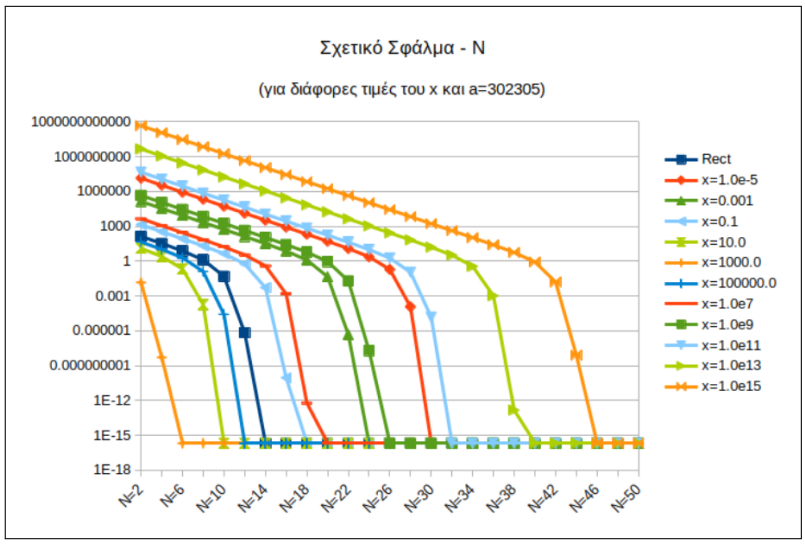
```
print('\n')
for x in x_range
print("x=", x, ", ")
for N in N_range
print(err_NR(a, N, x), ", ")
end
print('\n')
end
return
end
```

Για την επεξεργασία ή απλά τη μεταφορά των δεδομένων, βολεύει να τα αποθηκεύσουμε σε ένα αρχείο (εδώ ο πιο φυσικός τύπος αρχείου είναι το comma separated values file, με κατάληξη csv). Αυτό μπορούμε να το κάνουμε τρέχοντας το αρχείο που περιέχει τη συνάρτηση στη γραμμή εντολών με τον ακόλουθο τρόπο:

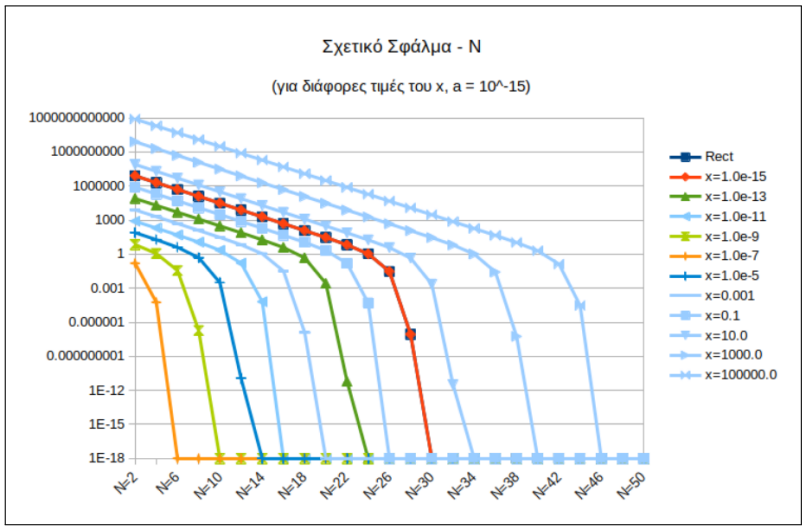
```
path_to_file1> julia file1.jl > file2.csv
```

Εδώ ο τελεστής > που προστέθηκε μετά από τη συνηθισμένη εντολή εκτέλεσης του προγράμματος δηλώνει πως η εκτύπωση των δεδομένων θα πραγματοποιείται όχι στο παράθυρο της γραμμής εντολών στην οθόνη, αλλά εντός του αρχείου csv.

Κατά την πρώτη εκτέλεση της συνάρτησης, η τιμή του αριθμού a του οποίου προσεγγίστηκε η ρίζα ήταν 302.305, ενώ η αρχική προσέγγιση x έλαβε ως τιμές διαδοχικές δυνάμεις του 10. Για κάθε ένα από αυτά τα ζεύγη τιμών των a και x, το N πήρε τιμές από το 2 μέχρι το 50, με βήμα 2. Τα αποτελέσματα παρουσιάζονται στο ακόλουθο διάγραμμα:



Προκειμένου να μην υπάρχει πλήρης εξάρτηση των αποτελεσμάτων από μία μόνο τιμή του a, η παραπάνω διαδικασία επαναλήφθηκε για $a = 10^{-15}$:



Παρατηρούμε ότι σε όλες τις περιπτώσεις, το ποια είναι η μέθοδος με το μικρότερο σφάλμα δεν επηρεάζεται από τον αριθμό των επαναλήψεων (η γραμμή που αντιστοιχεί στη μέθοδο Rect δεν διασταυρώντεται με κάποια άλλη γραμμή). Επομένως, η μέθοδος της οποίας το αρχικό σχετικό σφάλμα είναι μικρότερο (δηλαδή αυτή της οποίας η αρχική προσέγγιση, που είναι ίση με x για τη Newton-Raphson και με 1 για τη μέθοδο των ορθογωνίων, είναι πλησιέστερη στην πραγματική τιμή της ρίζας), είναι αυτή που έχει το μικρότερο σφάλμα ύστερα από οποιονδήποτε αριθμό επαναλήψεων. Δηλαδή, φαίνεται ότι η μέθοδος Newton-

Raphson έχει μικρότερο σχετικό σφάλμα (ύστερα από οποιονδήποτε αριθμό επαναλήψεων) αν και μόνο αν η τιμή του x που της παρέχεται είναι καλύτερη προσέγγιση της ρίζας απ' ό,τι το 1.

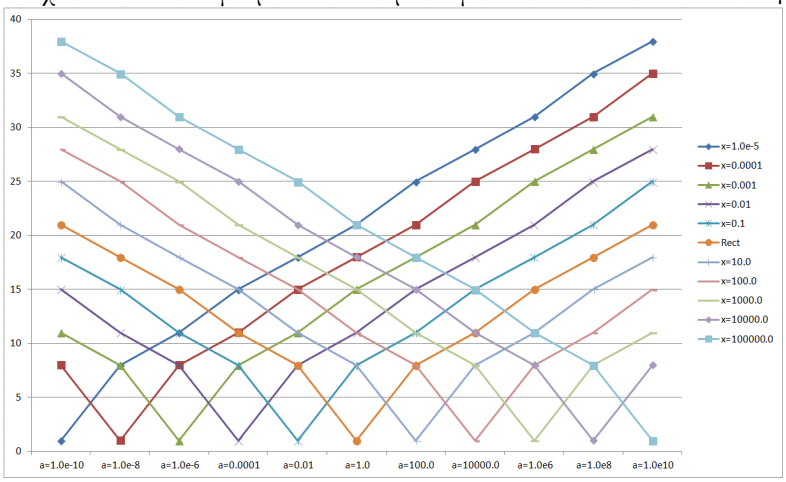
Επειδή, όμως, τα παραπάνω δεδομένα αφορούν δύο μόνο συγκεκριμένες τιμές του a, καλό είναι να ενισχύσουμε το τελευταίο μας συμπέρασμα με επιπλέον δεδομένα. Συγκεκριμένα, θα ελέγξουμε για διάφορες τιμές του a (και διάφορες σταθεροποιημένες τιμές του x) εάν όντως η μέθοδος της οποίας το σφάλμα ελαχιστοποιείται ύστερα από μικρότερο αριθμό επαναλήψεων είναι, όπως αναμένουμε, αυτή στην οποία παρέχεται καλύτερη αρχική προσέγγιση της ρίζας. Για την παραγωγή αυτών των δεδομένων κατασκευάζουμε δύο συναρτήσεις που επιστρέφουν τον αριθμό των επαναλήψεων που απαιτούνται για την ελαχιστοποίηση και ύστερα μία ακόμα που καλεί τις προηγούμενες:

```
function high_accur_rect(a, relerr)
sqa = sqrt(a)
N = 1
while abs(rect(a, N) - sqa)/sqa > relerr
N+=1
end
return N
end
```

```
function high_accur_NR(a, x, relerr)
sqa = sqrt(a)
N = 1
while abs(NewtonRaphson(a, N, x) - sqa)/sqa > relerr
N+=1
end
return N
end
```

```
function err_comp_a_x(a_range, x_range, err)
print(" , ")
for a in a_range
print("a=", a, ", ")
end
print('\n')
print("Rect", ", ")
for a in a_range
print(high_accur_rect(a, err), ", ")
end
print('\n')
for x in x_range
print("x=", x, ", ")
for a in a_range
print(high_accur_NR(a, x, err), ", ")
end
print('\n')
end
return
end
```

Κατά την εκτέλεση της τελευταίας συνάρτησης, η παράμετρος a έλαβε ως τιμές δυνάμεις του 10, με τον εκθέτη να καλύπτει τις ακέραιες τιμές από -10 μέχρι και 10 με βήμα 2, ενώ η παράμετρος x έλαβε ως τιμές διαδοχικές δυνάμεις του 10, από -5 μέχρι 5. Η παράμετρος err έλαβε την τιμή $2.08 \cdot 10^{-16}$, ώστε να επιτευχθεί το ελάχιστο δυνατό σφάλμα. Τα δεδομένα φαίνονται στο ακόλουθο διάγραμμα:



Όπως αναμέναμε, το σχετικό σφάλμα ελαχιστοποιείται ύστερα από μικρότερο αριθμό επαναλήψεων με τη Newton-Raphson απ' ό,τι με τη μέθοδο των ορθογωνίων όταν στην πρώτη παρέχεται προσέγγιση της ρίζας που είναι καλύτερη από το 1. Άρα φαίνεται πως πράγματι αυτή είναι η ικανή και αναγκαία συνθήκη ώστε η μέθοδος Newton-Raphson να έχει μικρότερο σχετικό σφάλμα και σε όλες τις

ενδιάμεσες τιμές του N, για οποιεσδήποτε τιμές των a και x.

6 Σύγκριση ως προς τον χρόνο εκτέλεσης N επαναλήψεων

Και πάλι, όπως στη σύγκριση ως προς το σχετικό σφάλμα, οι μόνες παράμετροι που ενδέχεται να επηρεάζουν το ποια μέθοδος απαιτεί λιγότερο χρόνο για την πραγματοποίηση ορισμένου αριθμού επαναλήψεων είναι ο αριθμός N των επαναλήψεων, η αρχική προσέγγιση x που παρέχεται στον αλγόριθμο Newton-Raphson και ο αριθμός a του οποίου προσεγγίζουμε τη ρίζα. Ως μέτρο σύγκρισης των δύο αλγορίθμων μπορούμε να υιοθετήσουμε τον λόγο των χρόνων εκτέλεσης N επαναλήψεων.

Για την παραγωγή των δεδομένων, τα οποία θέλουμε να έχουν τη μορφή χρόνων εκτέλεσης, χρησιμοποιούμε τις ακόλουθες συναρτήσεις:

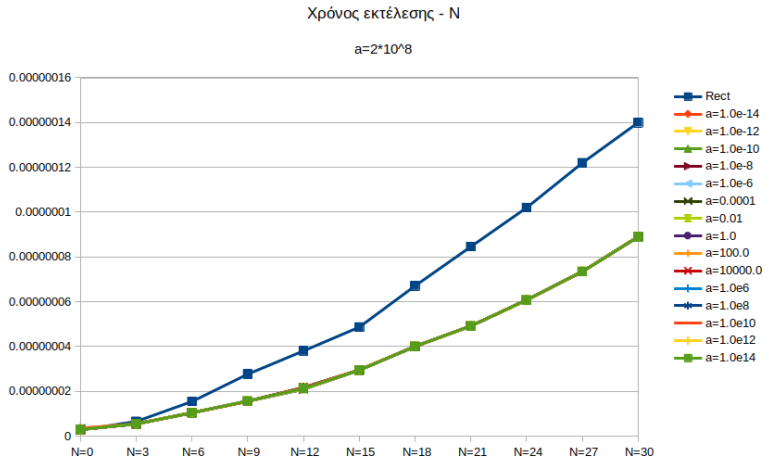
```
using BenchmarkTools

function time_rect(a, N)
t = @belapsed rect($a, $N)
return t
end
```

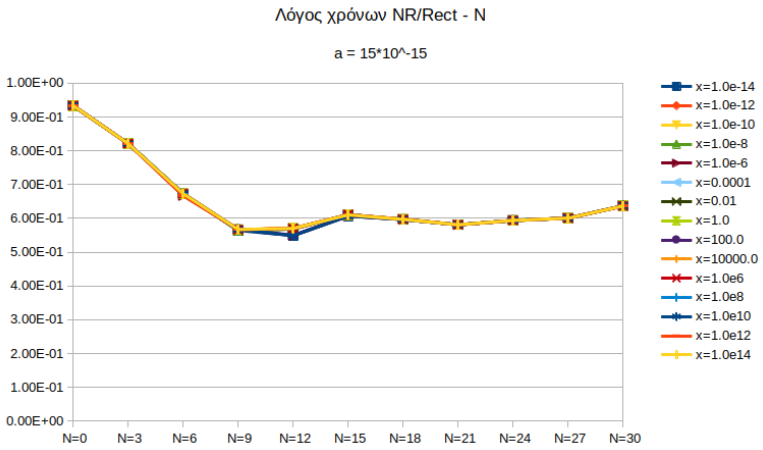
```
function time_NR(a, N, x)
t = @belapsed NewtonRaphson($a, $N, $x)
return t
end
```

Επιλέγουμε να διερευνήσουμε τους παράγοντες με τη σειρά που τους παρουσιάσαμε παραπάνω, επομένως εστιάζουμε αρχικά την προσοχή μας στον αριθμό των επαναλήψεων, N. Σκοπεύουμε να καλέσουμε διαδοχικά τις παραπάνω συναρτήσεις για σταθεροποιημένες τιμές των x, a και για διάφορες τιμές του N. Τα δεδομένα που θα προκύψουν θα έχουν τη μορφή ενός $2 \times n$ πίνακα, όπου n είναι το πλήθος των τιμών που λαμβάνει η παράμετρος N (θα υπάρχει μία γραμμή με τα αποτελέσματα της πρώτης συνάρτησης και μία γραμμή με τα αποτελέσματα της δεύτερης, για την τιμή της αρχικής προσέγγισης που έχουμε σταθεροποιήσει). Ωστόσο, προκειμένου να μην υπάρχει πλήρης εξάρτηση από μία τιμή του x, θα μπορούσαμε να προσθέσουμε και άλλες γραμμές, κάθε μία από τις οποίες θα αντιπροσωπεύει μία τιμή της συγκεκριμένης παραμέτρου. Με το σκεπτικό αυτό, κατασκευάζουμε την ακόλουθη συνάρτηση:

```
function err_comp(a, N_range, x_range)
print(", ")
for N in N_range
print("N=", N, ", ")
end
print('\n', "Rect, ")
for N in N_range
print(err_rect(a, N), ", ")
end
print('\n')
for x in x_range
print("x=", x, ", ")
for N in N_range
print(err_NR(a, N, x), ", ")
end
print('\n')
end
return
end
```

Το παραπάνω πρόγραμμα εκτελέστηκε αρχικά για $a = 2 \cdot 10^8$. Το x πήρε ως τιμές δυνάμεις του 10, με τον εκθέτη να λαμβάνει τιμές από το -14 μέχρι και το $+14$, με βήμα 2, και το N να λαμβάνει τιμές από 0 μέχρι 30, με βήμα 3. Ύστερα πραγματοποιήθηκε μια επανεκτέλεση για $a = 15 \cdot 10^{-15}$. Τέλος, τα δεδομένα επεξεργάστηκαν έτσι ώστε να προκύψουν οι λόγοι των χρόνων των δύο μεθόδων. Στο πρώτο από τα ακόλουθα διαγράμματα παρουσιάζουμε τα αρχικά δεδομένα, ενώ στο δεύτερο παρουσιάζουμε τους λόγους:





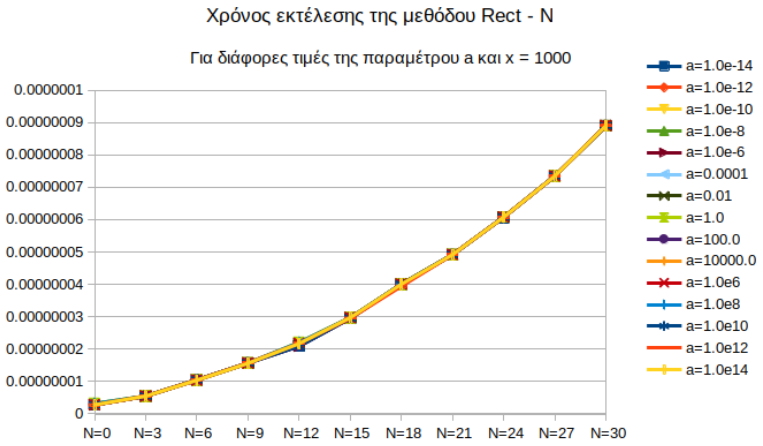
Παρατηρούμε ότι, με ελάχιστες εξαιρέσεις, όλες οι τιμές που αφορούν τη μέθοδο Newton-Raphson συμπέφτουν, για όλες τις τιμές του x. Επομένως, μπορούμε να συγκρίνουμε τις δύο μεθόδους χωρίς να λαμβάνουμε υπόψη την αρχική προσέγγιση. Στο πρώτο διάγραμμα, λοιπόν, παρατηρούμε ότι η μέθοδος των ορθογωνίων απαιτεί περισσότερο χρόνο και όμοια στο δεύτερο ο λόγος έχει γενικά τιμές μικρότερες από 1, το οποίο σημαίνει ότι η μέθοδος Newton-Raphson είναι πιο γρήγορη. Φαίνεται άρα, εκ πρώτης όψεως, ότι μάλλον το ποια μέθοδος απαιτεί λιγότερο χρόνο για N επαναλήψεις δεν εξαρτάται ούτε από την τιμή της παραμέτρου a, αλλά θα χρειαστούμε περισσότερες τιμές της προκειμένου να το επιβεβαιώσουμε.

Προκειμένου να εξακριβώσουμε, τώρα, τον τρόπο με τον οποίο επηρεάζει η συγκεκριμένη παράμετρος τους χρόνους εκτέλεσης, μπορούμε να βασιστούμε σε μια τροποποίηση της προηγούμενης συνάρτησης. Ουσιαστικά το μόνο που χρειάζεται να αλλάξουμε είναι οι ρόλοι των a και x, δηλαδή το εύρος τιμών x_range θα πρέπει να ερμηνεύεται από το πρόγραμμα ως το σύνολο των τιμών που θα λαμβάνει ο αριθμός του οποίου προσεγγίζεται η ρίζα, ενώ η σταθερά a θα πρέπει να ερμηνεύεται ως αρχική προσέγγιση. Αυτό μπορούμε να το πραγματοποιήσουμε αλλάζοντας τα ορίσματα των συναρτήσεων time_NR και time_rect (π.χ. το πρώτο όρισμα γίνεται x αντί για a). Ύστερα από αυτήν την αλλαγή και μερικές άλλες αναγκαίες διορθώσεις που αυτή επιφέρει[1] προκύπτει η ακόλουθη συνάρτηση:

[1] Συγκεκριμένα, η συνάρτηση time_rect, η οποία προηγουμένως καλούνταν για μία μόνο τιμή του αριθμού του οποίου προσεγγίζουμε τη ρίζα, τώρα θα πρέπει να μπει μέσα στο for loop που πραγματοποιούμε πάνω στις τιμές x_range, έτσι ώστε να χρησιμοποιηθεί για την προσέγγιση της ρίζας όλων των αριθμών που επιθυμούμε.

```
function time_comp_reverse(a, N_range, x_range)
print(", ")
for N in N_range
print("N=", N, ", ")
end
print('\n')
for x in x_range
print("a=", x, ", ")
for N in N_range
print(time_NR(x, N, a), ", ")#print(time_NR(a, N, x), ", ")
end
print('\n')
end
for x in x_range
print("a=", x, ", ")#print("x=", x, ", ")
for N in N_range
print(time_rect(x, N), ", ")
end
print('\n')
end
return end
```

Εκτελώντας την παραπάνω συνάρτηση με την αρχική προσέγγιση ίση με 1000, τον αριθμό του οποίου προσεγγίζουμε τη ρίζα να παίρνει ως τιμές δυνάμεις του 10 (με εκθέτη από -14 μέχρι 14 με βήμα 2) και το N να παίρνει τιμές ίδιες με προηγουμένως, προκύπτουν (αφού αποθηκεύσουμε τα δεδομένα σε αρχείο .csv και πραγματοποιήσουμε κατάλληλη επεξεργασία των δεδοδμένων) τα ακόλουθα διαγράμματα:



Παρατηρούμε ότι και για τις δύο μεθόδους, οι γραμμές ταυτίζονται, γεγονός που επιβεβαιώνει την ανεξαρτησία από το a. Συνεπώς, μπορούμε να πούμε γενικότερα (χωρίς κανέναν περιορισμό) ότι η μέθοδος Newton - Raphson είναι γρηγορότερη από τη μέθοδο των ορθογωνίων (και ο ακριβής λόγος των χρόνων εκτέλεσής τους εξαρτάται από το N, αλλά για αρκούντως μεγάλο πλήθως επαναλήψεων φαίνεται να σταθεροποιείται περίπου στο 0,75).

7 Συμπέρασμα

Τελικά, η μέθοδος Newton-Raphson έχει μικρότερο σφάλμα σε σχέση με τη μέθοδο των ορθογωνίων, ύστερα από οποιονδήποτε αριθμό επαναλήψεων, αν και μόνο αν της έχει δοθεί καλύτερη αρχική προσέγγιση από αυτήν που έχει δοθεί στη μέθοδο των ορθογωνίων, δηλαδή καλύτερη αρχική προσέγγιση από τη μονάδα. Επιπλέον, η μέθοδος Newton-Raphson, με την υλοποίηση σε περιβάλλον Julia που πραγματοποιήσαμε, ήταν ταχύτερη από τη μέθοδο των ορθογωνίων (ολοκλήρωσε οποιονδήποτε αριθμό επαναλήψεων σε μικρότερο χρόνο). Για τον λόγο αυτό, θα πρέπει να προτιμάται από τη μέθοδο των ορθογωνίων, υπό τον όρο να της παρέχεται η βέλτιστη ήδη υπάρχουσα προσέγγιση της ρίζας (δηλαδή είτε η μονάδα, που που είναι πάντα η αρχική προσέγγιση της μεθόδου των ορθογωνίων, ή κάποια ακόμα καλύτερη προσέγγιση της τετραγωνικής ρίζας).

ΠΑΡΑΡΤΗΜΑ: Σύγκριση με άλλες γλώσσες προγραμματισμού

Σε αυτό το παράρτημα θα συγκρίνουμε την Julia με τις γλώσσες MATLAB και Python ως προς την ταχύτητα εκτέλεσης των δύο μεθόδων (Newton Raphson και Rect). Για τον σκοπό αυτό, θα μετατρέψουμε τον κώδικα Julia που χρειαστήκαμε για να μετρήσουμε τον χρόνο εκτέλεσης (συγκεκριμένα τις συναρτήσεις Newton-Raphson, rect, time_NR, time_rect και time_comp) σε αντίστοιχο κώδικα των άλλων δύο γλωσσών. Εκτελώντας τις νέες αυτές συναρτήσεις, θα αποκτήσουμε τους χρόνους εκτέλεσης των ίδιων πράξεων με τις γλώσσες MATLAB και Python, οπότε μετά θα μπορέσουμε να τους συγκρίνουμε με αυτούς της Julia. Ξεκινάμε λοιπόν από την υλοποίηση σε MATLAB R2021b:

```
function W = rect(a, N)
L = a;
W = 1;
for i=1:N
W = (L+W)/2;
L = a/W;
end
end
```

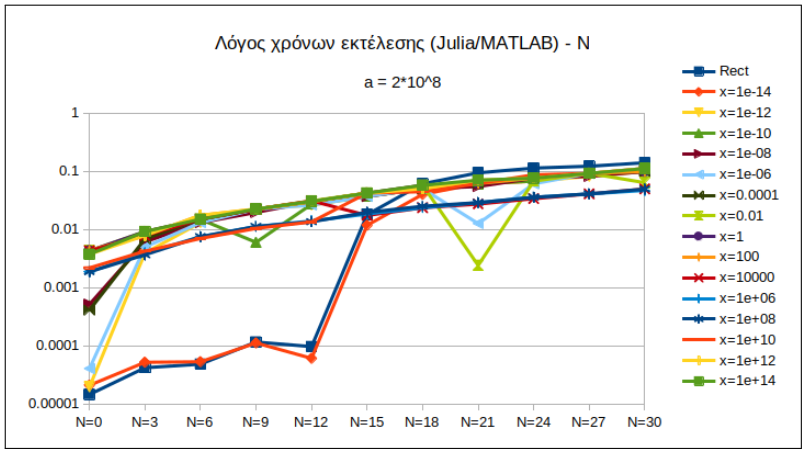
```
function sqa = NewtonRaphson(a, N, x)
sqa = x;
for i=1:N
sqa = (sqa + a/sqa)/2;
end
end
```

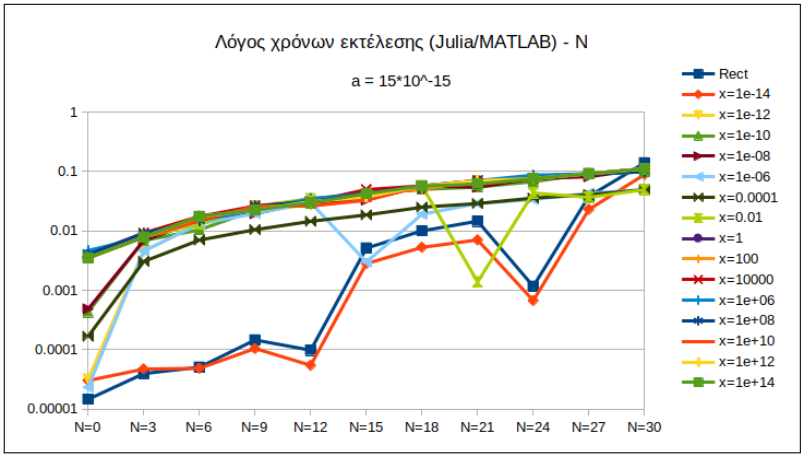
```
function t = time_rect(a, N)
tic
rect(a, N);
t = toc;
end
```

```
function t = time_NR(a, N, x)
tic
NewtonRaphson(a, N, x);
t = toc;
end
```

```
function time_comp(a, N_range, x_range)
f = fopen("NR_matlab.txt", 'a');
fprintf(f, ");
l = length(N_range);
for i = 1:l
fprintf(f, "N=%g,", N_range(i));
end
fprintf(f, "\nRect, ");
for i = 1:l
fprintf(f, "%g, ", time_rect(a, N_range(i)));
end
fprintf(f, '\n');
for i = 1:length(x_range)
x = x_range(i);
fprintf(f, "x=%g, ", x);
for j = 1:l
fprintf(f, "%g, ", time_NR(a, N_range(j), x));
end
fprintf(f, '\n');
end
fclose(f);
end
```

Ας παρατηρήσουμε ότι εδώ για την αποθήκευση των δεδομένων χρησιμοποιήθηκε η κατάληξη .txt αντί για .csv, αλλά και πάλι ήταν δυνατό το άνοιγμα του αρχείου σε κάποιο υπολογιστικό φύλλο (π.χ. με πρόγραμμα Microsoft Office Excel) για την περαιτέρω επεξεργασία τους. Έτσι, τα δεδομένα που προέκυψαν από τις δύο εκτελέσεις της συνάρτησης time_comp (μία με $a = 2 \cdot 10^8$ και μία με $a = 15 \cdot 10^{-15}$) συγκρίθηκαν με τα αντίστοιχά τους από τη γλώσσα Julia και δημιουργήθηκαν τα ακόλουθα διαγράμματα:





Αγνοώντας τις πρώτες εκτελέσεις, όπου πιθανότατα ο χρόνος που επέστρεψε η συνάρτηση toc της MATLAB είναι υψηλότερος από τον πραγματικό και δεν θα πρέπει να ληφθεί υπόψη (όπως, κατ' αναλογία, η πρώτη εκτέλεση της εντολής @time στη Julia δεν πρέπει να λαμβάνεται υπόψη), βλέπουμε ότι ο λόγος παίρνει τιμές κοντά στο 0.1, δηλαδή η Julia φαίνεται πως είναι περίπου 10 φορές γρηγορότερη από τη MATLAB.

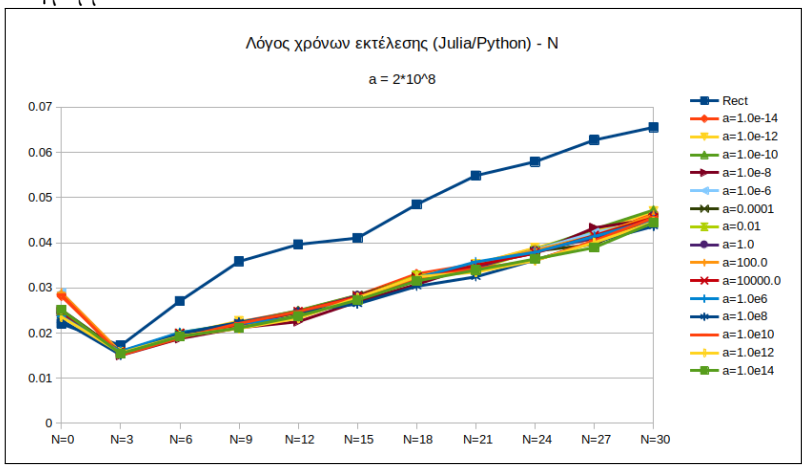
Επαναλαμβάνουμε τώρα την ίδια διαδικασία με την Python (version 3.10.6), ξεκινώντας από τις συναρτήσεις. Η πιο αξιόπιστη συνάρτηση που βρέθηκε για τη μέτρηση του χρόνου εκτέλεσης είναι η timeit, της ομώνυμης βιβλιοθήκης:

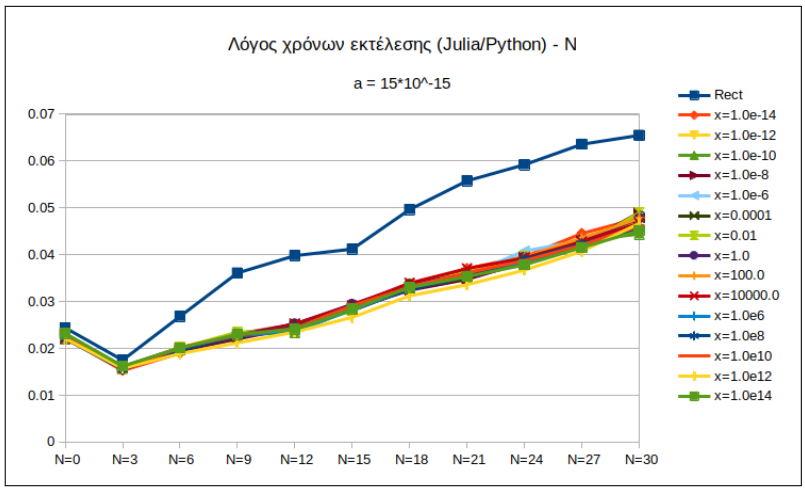
```
import timeit


rect = """
L = a
W = 1
for i in range(1, N+1):
    W = (L+W)/2
    L = a/W
"""


def time_rect(a, N):
    t = timeit.timeit(rect, number = 10000, globals = locals())
    return t/10000
```

```
import timeit


NewtonRaphson = """
sqa = x
for i in range(1, N+1):
    sqa = (sqa + a/sqa)/2
"""


def time_NR(a, N, x):
    t = timeit.timeit(NewtonRaphson, number = 10000, globals = locals())
    return t/10000
```

```
def time_comp(a, N_range, x_range):
    print(", ", end = ")
    for N in N_range:
        print("N=", N, ", ", sep = "", end = "")
    print('\n', "Rect, ", sep = "", end = "")
    for N in N_range:
        print(time_rect(a, N), ", ", sep = "", end = "")
    print('\n', end = "")
    for x in x_range:
        print("x=", x, ", ", sep = "", end = "")
        for N in N_range:
            print(time_NR(x, N, a), ", ", sep = "", end = "")
        print('\n', end = "")
```

Εκτελώντας και πάλι την τελευταία συνάρτηση δύο φορές, προέκυψαν τα ακόλουθα διαγράμματα:





Παρατηρούμε ότι, με βάση το εύρος των τιμών που λαμβάνει ο λόγος, η Julia εκτελεί τις μεθόδους περίπου από 15 έως 75 φορές γρηγορότερα σε σύγκριση με την Python.

Σημείωση: Επειδή αναγκαστικά για τη μέτρηση των χρόνων εκτέλεσης των μεθόδων σε κάθε γλώσσα χρησιμοποιήθηκαν διαφορετικές εντολές, ένα μικρό μέρος της διαφοράς στους χρόνους εκτέλεσης μπορεί να οφείλεται στις διαφορετικές μεθόδους προσδιορισμού τους. Για παράδειγμα, στην Python ο χρόνος εκτέλεσης θεωρήθηκε ότι είναι ο μέσος όρος των επιμέρους χρόνων 10.000 εκτελέσεων, ενώ στην Julia ο χρόνος εκτέλεσης ήταν ίσος με τον ελάχιστο από τους επιμέρους χρόνους (και στη MATLAB δεν υπήρχαν καθόλου επιμέρους χρόνοι, αφού πραγματοποιούνταν μία μόνο εκτέλεση και ο χρόνος της καταχωρούνταν κατ' ευθείαν στο αρχείο).