Gradien Based Optimization

Μέθοδοι Βελτιστοποίησης
Δεύτερη Υποχρεωτική Εργασία

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών

Ονοματεπώνυμο: Παπαδάκης Μιχαήλ
Αριθμός Μητρώου: 02118026
Ακαδημαϊκό έτος: 3^o
Ημερομηνία: 11/1/22

$\boxed{\text{K1} = 6}$

Περιεχόμενα

Κατάλογος Πινάκων

Κατάλογος Σχημάτων

1. Εισαγωγή

Στην εργασία γίνεται εύρεση παραγώγων με διάφορες τεχνικές. Υπάρχουν δυο προβλήματα, ένα με σκοπό να κατανοηθούν οι βασικές τεχνικές παραγώγισης (direct differentiation, continuous and discrete adjoint) και ένα δεύτερο πιο απλό με σκοπό να εξερευνηθούν άλλες τεχνικές παραγώγισης (complex differentiation, algorithmic-automatic differentiation).

2. Τμήμα 2.1

Το αντικείμενο του πρώτου μέρους είναι ένα πρόβλημα αντίστροφου σχεδιασμού, οπού το ζητούμενο είναι το ύψος του οριακού στρώματος σε μια επίπεδη πλάκα, σε μια ασυμπίεστη ροή με σταθερή πίεση κατά μήκος του άξονα x, που δίνεται από τον τύπο 1 με την οριακή συνθήκη 2, να πλησιάσει όσο περισσότερο γίνεται μια επιθυμητή κατανομή που δίνεται από τον τύπο 3.

$$\delta \frac{d\delta}{dx} = \frac{\pi^2 v}{(4-\pi)u_e} \tag{1}$$

$$\delta|_{x=0} = \delta_o = 2 + K1 = 2 + 6 = 8 \ [mm] \tag{2}$$

$$\delta_{target} = \delta_o(1 + 10x - 5x^2) \tag{3}$$

Οι ελεύθερες μεταβλητές (design variables) του προβλήματος είναι η συνεκτικότητα του ρευστού v, και η ταχύτητα στα όρια του οριακού στρώματος u_e η οποία είναι σταθερή κατά x. Η αντικειμενική συνάρτηση, η οποία πρέπει να ελαχιστοποιηθεί ορίστηκε ως[1]:

$$F = \int_0^1 (\delta - \delta_{target})^2 dx \tag{4}$$

Το πρώτο βήμα για την επίλυση του παραπάνω προβλήματος βελτιστοποίησης είναι η λύση του primal προβλήματος.

2.1 Τμήμα 2.1.α - Διακριτοποίηση χωρίου και επίλυση primal

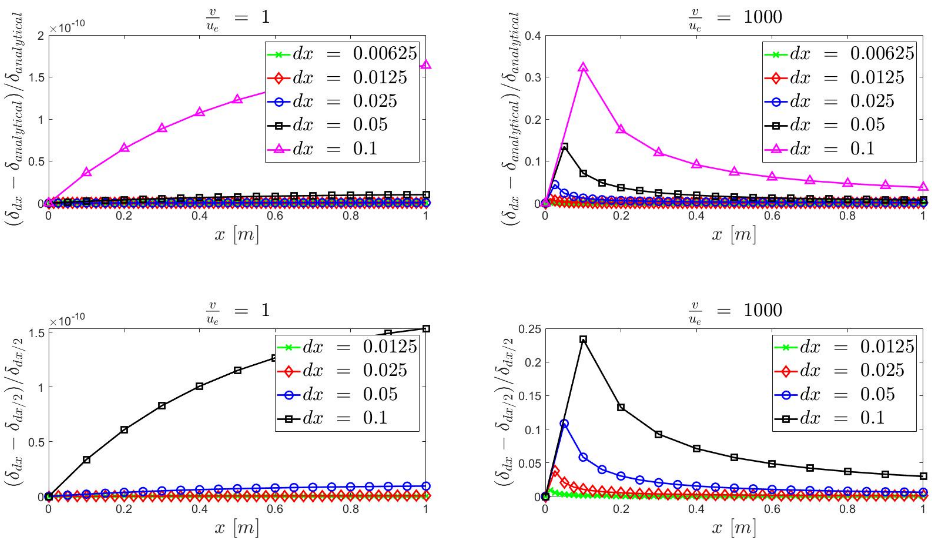
Για την επίλυση της εξίσωσης 1 επιλέχθηκε η μέθοδος Runge-Kutta τέταρτης τάξης. Βέβαια, η λύση μπορεί να προκύψει και με αναλυτικό τρόπο, ο οποίος **θα παρατεθεί απλώς για λόγους σύγκρισης καθώς σε πραγματικά προβλήματα δεν θα υπάρχει η αναλυτική λύση του πρωτεύοντος προβλήματος**. Η αναλυτική λύση είναι:

$$\delta = \sqrt{\frac{2\pi^2 v}{(4-\pi)u_e} x + \delta_o^2} \tag{5}$$

Ακολουθεί η διερεύνηση της διακριτοποίησης για την επίλυση του πρωτεύοντος προβλήματος ώστε τα αποτελέσματα να είναι ικανοποιητικής ακρίβειας. Καταρχάς, παρατηρήθηκε πως ανάλογα με τον λόγο v/u_e μπορεί να απαιτείται πυκνότερη διακριτοποίηση. Για αυτό τον λόγο, η σύγκριση διαφορετικών διακριτοποιήσεων γίνεται για 2 διαφορετικές τιμές του λόγου v/u_e, ώστε να ληφθεί βήμα που ανταποκρίνεται σε όλες τις περιπτώσεις. Ακολουθούν διαγράμματα που συγκρίνουν την αριθμητική λύση που προκύπτει με ένα βήμα dx με μια άλλη λύση που έχει το υποδιπλάσιο βήμα. Όσο η τιμή αυτή πάει στο 0, τόσο η λύση που προκύπτει είναι ανεξάρτητη της διακριτοποίησης. Μάλιστα, για να κανονικοποιηθεί το αποτέλεσμα, η διαφορά $\delta_{dx} - \delta_{dx/2}$ διαιρείται με το $\delta_{dx/2}$. Στο ίδιο σχήμα υπάρχει και η σύγκριση με την αναλυτική λύση για λόγους εποπτείας. **Στην πραγματικότητα στη διάθεση μας θα είναι μόνο το σχετικό σφάλμα ανάλογα με την διακριτοποίηση.**

Από το σχήμα 1 παρατηρείται πως με βήμα 0.0125 επιτυγχάνεται σχετικό σφάλμα μικρότερο του 1% το οποίο κρίνεται ικανοποιητικό. Μάλιστα, για μικρότερο λόγο v/u_e το σχετικό σφάλμα είναι αμελητέο. Ωστόσο, τελικά το βήμα επιλέχθηκε ως το μισό από αυτό, για να μπορέσει να υλοποιηθεί και η ευθεία διαφόριση. Αυτό θα εξηγηθεί και παρακάτω.

[1] Το τετράγωνο απορροφά τις διαφορές προσήμου, ενώ ο όρος dx απορροφά την επίδραση της διακριτοποίησης.



Σχήμα 1: Δοκιμές διακριτοποίησης για επιλογή βήματος runge-kutta

2.2 Τμήμα 2.1.β - Εύρεση παραγώγων

2.2.1 Ευθεία διαφόριση

Στην ευθεία διαφόριση αρχικά διαφορίζεται η αντικειμενική συνάρτηση:

$$\frac{dF}{db_i} = \frac{d}{db_i}\int_0^1 (\delta - \delta_{target})^2 dx = \int_0^1 2(\delta - \delta_{target})\frac{d\delta}{db_i}dx \tag{6}$$

Συνεπώς, πρέπει να βρεθεί η παράγωγος $\frac{d\delta}{db_i}$. Έτσι, παραγωγίζονται οι σχέσεις του πρωτεύοντος προβλήματος (Αφού μετασχηματιστεί στη σχέση 7) οι οποίες παρουσιάζονται στις σχέσεις 8,9.

$$\frac{d\delta}{dx} = \frac{\pi^2 v}{(4-\pi)u_e}\frac{1}{\delta} \tag{7}$$

Διαφορίζοντας προκύπτει:

$$\frac{d}{db_i}(\frac{d\delta}{dx}) = \frac{d}{db_i}(\frac{\pi^2 v}{(4-\pi)u_e}\frac{1}{\delta}) = \frac{d}{db_i}(\frac{\pi^2 v}{(4-\pi)u_e})\frac{1}{\delta} + \frac{\pi^2 v}{(4-\pi)u_e}\frac{d}{db_i}(\frac{1}{\delta})$$

Επειδή η μεταβλητή x δεν εξαρτάται από τις μεταβλητές σχεδιασμού, μπορεί να αλλάξει η σειρά διαφόρισης στο αριστερό μέλος. Έτσι, προκύπτουν οι σχέσεις:

$$\frac{d}{dx}(\frac{d\delta}{db_1}) = -\frac{\pi^2 v}{(4-\pi)u_e}\frac{1}{\delta^2}\frac{d\delta}{db_1} + \frac{\pi^2}{(4-\pi)u_e}\frac{1}{\delta} \tag{8}$$

$$\frac{d}{dx}(\frac{d\delta}{db_2}) = -\frac{\pi^2 v}{(4-\pi)u_e}\frac{1}{\delta^2}\frac{d\delta}{db_2} - \frac{\pi^2 v}{(4-\pi)u_e^2}\frac{1}{\delta} \tag{9}$$

Θεωρώντας σαν αγνώστους τις μεταβλητές $\frac{d\delta}{db_i}$, οι σχέσεις 8,9 είναι δυο γραμμικές διαφορικές εξισώσεις οι μάλιστα είναι απεμπλεγμένες και συνεπώς λύνονται ξεχωριστά. Σαν οριακή συνθήκη στις παραπάνω διαφορικές πάρθηκε:

$$\left.\frac{d\delta}{db_i}\right|_{x=0} = 0 \tag{10}$$

επειδή $\delta(0) = \delta_o$ ανεξάρτητα από την επιλογή των μεταβλητών σχεδιασμού. Χρησιμοποιείται ξανά η μέθοδος Runge-Kutta 4ης τάξης, σε μια παραλλαγή, καθώς η συνάρτηση δέχεται και τα $\delta(x)$ τα οποία θεωρούνται γνωστά κατά την επίλυση. Καθώς όμως στην Runge-Kutta πρέπει κατά την διαδικασία να βρεθούν τιμές ενδιάμεσα από τους κόμβους (δηλαδή οι τιμές $\delta_{i+dx/2}$)

πρέπει είτε να χρησιμοποιηθεί η 1 ώστε σε κάθε βήμα να υπολογίζονται οι ενδιάμεσες τιμές του πάχους, είτε το βήμα επίλυσης των διαφορικών 8,9 να είναι το διπλάσιο του βήματος επίλυσης του primal . Εδώ χρησιμοποιήθηκε η δεύτερη προσέγγιση.

Αφού επιλυθούν οι παραπάνω σχέσεις, προκύπτουν τα $\frac{d\delta}{db_i}(x)$. Έτσι, επιστρέφοντας στη σχέση 6 μπορεί να γίνει ο υπολογισμός των παραγώγων ευαισθησίας. Χρησιμοποιείται η μέθοδος simpson 1/3 για την ολοκλήρωση.

2.2.2 Συνεχή συζυγής μέθοδος

Αρχικά, η αντικειμενική γράφεται ως:

$$F = \int_0^1 (\delta - \delta_{target})^2 dx + \int_0^1 \Psi(\delta \frac{d\delta}{dx} - \underbrace{\frac{\pi^2 v}{(4-\pi)u_e}}_{K})dx \tag{11}$$

Το δεύτερο ολοκλήρωμα ισούται με μηδέν λόγω της 1. Συνεπώς, δεν αλλάζει η τιμή της αντικειμενικής! Παραγωγίζοντας την παραπάνω σχέση:

$$\frac{dF}{db_i} = \int_0^1 2(\delta - \delta_{target})\frac{d\delta}{db_i}dx + \underbrace{\int_0^1 \Psi \frac{d}{db_i}(\delta \frac{d\delta}{dx} - K)dx}_{T1} + \int_0^1 \frac{d\Psi}{db_i} \underbrace{(\delta \frac{d\delta}{dx} - \frac{\pi^2 v}{(4-\pi)u_e})}_{=0} dx \tag{12}$$

Κάνοντας τις πράξεις μόνο στο $T1$ προκύπτει:

$$T1 = \int_0^1 \Psi(\frac{d\delta}{db_i}\frac{d\delta}{dx} + \delta \frac{d}{db_i}\frac{d\delta}{dx} - \frac{dK}{db_i})dx$$

$$T1 = \int_0^1 \Psi \frac{d\delta}{dx}\frac{d\delta}{db_i}dx - \int_0^1 \Psi \frac{dK}{db_i}dx + \underbrace{\int_0^1 \Psi\delta \frac{d}{dx}\frac{d\delta}{db_i}dx}_{T2}$$

Η αλλαγή σειράς παραγώγισης στο τελευταίο ολοκλήρωμα γίνεται επειδή η μεταβλητή x δεν εξαρτάται από τις μεταβλητές σχεδιασμού. Με παραγοντική ολοκλήρωση, ο όρος Τ2:

$$T2 = \Psi\delta \left. \frac{d\delta}{db_i}\right|_0^1 - \int_0^1 \frac{d\Psi\delta}{dx}\frac{d\delta}{db_i}dx$$

$$T2 = \underbrace{\Psi\delta \left.\frac{d\delta}{db_i}\right|_0}_{=0} - \Psi\delta \left.\frac{d\delta}{db_i}\right|_1 - \int_0^1 (\frac{d\Psi}{dx}\delta + \Psi\frac{d\delta}{dx})\frac{d\delta}{db_i}dx$$

Ο πρώτος όρος είναι μηδέν καθώς $\delta(0) = \delta_o$ **ανεξάρτητα από την επιλογή των μεταβλητών σχεδιασμού**.Συνεπώς, ο όρος $T1$ γίνεται:

$$T1 = \int_0^1 \frac{d\delta}{db_i}(\Psi\frac{d\delta}{dx} - \frac{d\Psi}{dx}\delta - \Psi\frac{d\delta}{dx})dx - \int_0^1 \Psi \frac{dK}{db_i}dx - \Psi\delta \left.\frac{d\delta}{db_i}\right|_1$$

$$T1 = \int_0^1 \frac{d\delta}{db_i}(-\frac{d\Psi}{dx}\delta)dx - \int_0^1 \Psi \frac{dK}{db_i}dx - \Psi\delta \left.\frac{d\delta}{db_i}\right|_1$$

Οπότε η αρχική σχέση γίνεται:

$$\frac{dF}{db_i} = \int_0^1 2(\delta - \delta_{target})\frac{d\delta}{db_i}dx + \int_0^1 \frac{d\delta}{db_i}(-\frac{d\Psi}{dx}\delta)dx - \int_0^1 \Psi\frac{dK}{db_i}dx - \Psi\delta \left.\frac{d\delta}{db_i}\right|_1$$

$$\frac{dF}{db_i} = \underbrace{\int_0^1 \frac{d\delta}{db_i}(2(\delta - \delta_{target}) - \frac{d\Psi}{dx}\delta)dx}_{A} - \underbrace{\int_0^1 \Psi\frac{dK}{db_i}dx}_{B} - \underbrace{\Psi\delta \left.\frac{d\delta}{db_i}\right|_1}_{C} \tag{13}$$

Σκοπός της adjoint είναι η αποφυγή υπολογισμού της $\frac{d\delta}{db_i}$, και συνεπώς προκύπτει πως το Ψ πρέπει να επιλεγεί ώστε οι όροι A,C να μηδενίζονται. Έτσι, προκύπτει η Field Adjoint Equation

(FAE) που φαίνεται στη σχέση 14 με οριακές συνθήκες Adjoint Boundary Conditions που φαίνονται στη σχέση 15.

$$\frac{d\Psi}{dx} = \frac{2(\delta - \delta_{target})}{\delta} \tag{14}$$

$$\Psi|_1 = 0 \tag{15}$$

Καθώς η σχέση 14 στο δεξί μέρος είναι συνάρτηση μόνο του $\delta(x)$, η λύση της διαφορικής προκύπτει με ολοκλήρωση. Για την υλοποίησή της χρησιμοποιείται η συνάρτηση cumtrapz της matlab. Η συνάρτηση αυτή χρησιμοποιεί τη μέθοδο τραπεζίου και βρίσκει την τιμή της Ψ για κάθε x_i ολοκληρώνοντας μέχρι εκείνο το x_i. Η συνάρτηση cumtrapz δίνει $\Psi(0) = 0$, και $\Psi(1) \neq 0$. Για να εφαρμόσουμε την οριακή συνθήκη 15, αφαιρείται το $\Psi(1)$.

Έχοντας πια λύσει την FAE , και επιστρέφοντας στη σχέση 13, μπορούν να υπολογιστούν οι παράγωγοι ευαισθησίας, χρησιμοποιώντας τον όρο B, καθώς είναι διαθέσιμο το $\Psi(x)$, και η παράγωγος $\frac{dK}{db_i}$ υπολογίζεται αναλυτικά. Τελικά, οι παράγωγοι ευαισθησίας δίνονται από τον τύπο 16. Τα αποτελέσματα θα παρατεθούν μαζί με τις άλλες μεθόδους για απευθείας σύγκριση.

$$\frac{dF}{d\boldsymbol{b}} = -\int_0^1 \Psi(x) \left\{ \tfrac{\pi^2}{(4-\pi)u_e}, \quad -\tfrac{\pi^2 v}{(4-\pi)u_e^2} \right\}^T dx \tag{16}$$

2.2.3 Διακριτή συζυγής μέθοδος

Στην διακριτή μέθοδο, το adjoint γίνεται αφού έχει γίνει πρώτα η διακριτοποίηση του χωρίου για την αριθμητική λύση. Έτσι, η μέθοδο ξεκινάει από μια εξίσωση της μορφής (primal):

$$\boldsymbol{R}(\boldsymbol{U}, \boldsymbol{b}) = 0 \tag{17}$$

Στο συγκεκριμένο πρόβλημα, καθώς το primal λύνεται με τη μέθοδο Runge-Kutta , η παραπάνω εξίσωση είναι η διακριτοποιημένη μορφή της εξίσωσης 7 (δηλαδή $f = \frac{\pi^2 v}{(4-\pi)u_e}\frac{1}{\delta}$) :

$$R_i = \delta_i - \underbrace{\left[\delta_{i-1} + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\right]}_{\mu_i} = 0, \ i > 1 \tag{18}$$

$$R_1 = \delta_1 - \delta_o = 0 \tag{19}$$

με

$$k_1 = \Delta x f(x_{i-1}, \delta_{i-1}) \ = \ \Delta x \frac{\pi^2 b_1}{(4-\pi)b_2}\frac{1}{\delta_{i-1}}$$

$$k_2 = \Delta x f(x_{i-1} + \Delta x/2, \delta_{i-1} + k_1/2) \ = \ \Delta x \frac{\pi^2 b_1}{(4-\pi)b_2}\frac{1}{(\delta_{i-1} + k_1/2)},$$

$$k_3 = \Delta x f(x_{i-1} + \Delta x/2, \delta_{i-1} + k_2/2) \ = \ \Delta x \frac{\pi^2 b_1}{(4-\pi)b_2}\frac{1}{(\delta_{i-1} + k_2/2)},$$

$$k_4 = \Delta x f(x_{i-1} + \Delta x, \delta_{i-1} + k_3) \ = \ \Delta x \frac{\pi^2 b_1}{(4-\pi)b_2}\frac{1}{(\delta_{i-1} + k_3)},$$

Σε πινακοποιημένη μορφή, η εξίσωση 18 γράφεται:

$$\begin{bmatrix} 1 & 0 & 0 & ... & 0 & 0 \\ \mu_1 & 1 & 0 & ... & 0 & 0 \\ 0 & \mu_2 & 1 & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & \mu_N & 1 \end{bmatrix} \cdot \begin{Bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \\ ... \\ \delta_N \end{Bmatrix} = \begin{Bmatrix} \delta_o \\ 0 \\ 0 \\ ... \\ 0 \end{Bmatrix} \tag{20}$$

Η επίλυση μπορεί να γίνει με από πάνω προς τα κάτω αντικατάσταση, και έτσι μπορούν να υπολογιστούν και οι συντελεστές μ_i, ο οποίοι βρίσκονται στο μητρώο. (Αυτό γίνεται στην πραγματικότητα στην Runge-Kutta). Παραγωγίζοντας την επαυξημένη συνάρτηση:

$$dF_{aug} = dF = \frac{\partial F}{\partial \boldsymbol{U}}\delta \boldsymbol{U} + \frac{\partial F}{\partial \boldsymbol{b}}\delta \boldsymbol{b} - \boldsymbol{\Psi}^T \underbrace{(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}}\delta \boldsymbol{U} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{b}}\delta \boldsymbol{b})}_{=0}$$

Ο τελευταίος όρος είναι μηδέν επειδή ισχύει $\boldsymbol{R} = 0$. Μαζεύοντας τους όρους :

$$dF = (\frac{\partial F}{\partial \boldsymbol{U}} - \boldsymbol{\Psi}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}})\delta \boldsymbol{U} + (\frac{\partial F}{\partial \boldsymbol{b}} - \boldsymbol{\Psi}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{b}})\delta \boldsymbol{b} \tag{21}$$

Στην ίδια φιλοσοφία με την συνεχή συζηγή μέθοδο, πρέπει να επιλεγεί $\boldsymbol{\Psi}$ τέτοιο, ώστε να μηδενισθεί ο πρώτος όρος της εξίσωσης 21. Έτσι, προκύπτει η Field Adjoint Equation .

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}}^T \boldsymbol{\Psi} = \frac{\partial F}{\partial \boldsymbol{U}}^T \tag{22}$$

Η εξίσωση αυτή είναι γραμμική ως προς $\boldsymbol{\Psi}$, καθώς το μητρώο $\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}}^T$ δεν εξαρτάται από το $\boldsymbol{\Psi}$. Για την επίλυσή της, αρχικά πρέπει να βρεθούν τα δυο μητρώα. Για την εύρεση του $\frac{\partial F}{\partial \boldsymbol{U}}^T$, χρησιμοποιείται η σχέση:

$$F = \int_0^1 (\delta - \delta_{target})^2 dx$$

Στην διακριτή μορφή γίνεται:

$$F = \sum_0^N (\delta_i - \delta_{target,i})^2 \Delta x$$

Και παραγωγίζοντας [2]:

$$\frac{\partial F}{\partial U_i} = \frac{\partial F}{\partial \delta_i} = 2(\delta_i - \delta_{target,i})\Delta x \tag{23}$$

Η εύρεση του $\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}}^T$ γίνεται παραγωγίζοντας το μητρώο $\boldsymbol{R}$, δηλαδή τη σχέση 18.

$$\frac{\partial R_i}{\partial U_i} = \frac{\partial R_i}{\partial \delta_i} = 1; \tag{24}$$

$$\frac{\partial R_i}{\partial U_{i-1}} = \frac{\partial R_i}{\partial \delta_{i-1}} = -1 - \frac{1}{6}\left[\frac{\partial \kappa_1}{\partial \delta_{i-1}} + 2\frac{\partial \kappa_2}{\partial \delta_{i-1}} + 2\frac{\partial \kappa_3}{\partial \delta_{i-1}} + \frac{\partial \kappa_4}{\partial \delta_{i-1}}\right] \tag{25}$$

όπου (ορίζοντας ως $const = \Delta x \frac{\pi^2}{(4-\pi)}$, $k_0 = 0$ και $\frac{\partial k_0}{\partial \delta_{i-1}} = 0$) οι παράγωγοι $\frac{\partial \kappa_j}{\partial \delta_{i-1}}$ δίνονται από τον τύπο :

$$\frac{\partial \kappa_j}{\partial \delta_{i-1}} = -\,const\frac{b1}{b2}\ \frac{1}{(\delta_{i-1} + k_{j-1}/2)^2}\ (1 + \frac{1}{2}\frac{\partial \kappa_{j-1}}{\partial \delta_{i-1}}),\quad \text{για } j = 2,3 \tag{26}$$

$$\frac{\partial \kappa_j}{\partial \delta_{i-1}} = -\,const\frac{b1}{b2}\ \frac{1}{(\delta_{i-1} + k_{j-1})^2}\ (1 + \frac{\partial \kappa_{j-1}}{\partial \delta_{i-1}}),\quad \text{για } j = 1,4 \tag{27}$$

Συνεπώς, χρησιμοποιώντας τις σχέσεις 24,25,26,27, σχηματίζεται ο πίνακας $\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}}^T$. Η μορφή του είναι:

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{U}}^T = \begin{bmatrix} 1 & \frac{\partial R_2}{\partial \delta_1} & 0 & ... & 0 & 0 \\ 0 & 1 & \frac{\partial R_3}{\partial \delta_2} & ... & 0 & 0 \\ 0 & 0 & 1 & ... & 0 & 0 \\ ... & ... & ... & ... & ... & \frac{\partial R_n}{\partial \delta_{n-1}} \\ 0 & 0 & 0 & ... & 0 & 1 \end{bmatrix}$$

Έτσι, η 22 λύνεται με από κάτω προς τα πάνω αντικατάσταση (αντίστοιχα με την ολοκλήρωση από το τέλος της συνεχής adjoint). Έχοντας βρει το διάνυσμα $\boldsymbol{\Psi}$, και γυρνώντας στη σχέση 21, μπορούν να υπολογιστούν οι παράγωγοι ευαισθησίας.

$$\frac{dF}{d\boldsymbol{b}} = \frac{\partial F}{\partial \boldsymbol{b}} - \boldsymbol{\Psi}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{b}} \tag{28}$$

[2]Για την διακριτή μέθοδο χρησιμοποιήθηκε η μέθοδο του τραπεζίου. Σε αυτή, ο τελευταίος όρος στο άθροισμα πολλαπλασιάζεται με $\Delta X/2$. Αυτό λήφθηκε υπόψη στον κώδικα.

Αρχικά, υπολογίζονται οι ποσότητες $\frac{\partial F}{\partial \boldsymbol{b}} = \mathbf{0}$ καθώς η αντικειμενική δεν εξαρτάται απευθείας από μεταβλητές σχεδιασμού και $\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{b}}$. Το δεύτερο, υπολογίζεται μαζί με τα υπόλοιπα κατά τη διάρκεια της επίλυσης. Από τη σχέση 18 προκύπτει:

$$\frac{\partial R_i}{\partial b_i} = 0 \tag{29}$$

$$\frac{\partial R_i}{\partial b_i} = -\frac{1}{6}\left[\frac{\partial \kappa_1}{\partial b_i} + 2\frac{\partial \kappa_2}{\partial b_i} + 2\frac{\partial \kappa_3}{\partial b_i} + \frac{\partial \kappa_4}{\partial b_i}\right] \tag{30}$$

όπου (ορίζοντας ως $const = \Delta x \frac{\pi^2}{(4-\pi)}$, $k_0 = 0$ και $\frac{\partial k_0}{\partial b_i} = 0$) οι παράγωγοι $\frac{\partial \kappa_j}{\partial b_i}$ δίνονται από τον τύπο :

$$\frac{\partial \kappa_j}{\partial b_1} = const\frac{1}{b_2}\frac{1}{(\delta_{i-1} + k_{j-1}/2)} - const\frac{b_1}{b_2}\frac{1}{(\delta_{i-1} + k_{j-1}/2)^2}\frac{1}{2}\frac{\partial \kappa_{j-1}}{\partial b_1}, \quad \text{για } j = 2,3 \tag{31}$$

$$\frac{\partial \kappa_j}{\partial b_1} = const\frac{1}{b_2}\frac{1}{(\delta_{i-1} + k_{j-1})} - const\frac{b_1}{b_2}\frac{1}{(\delta_{i-1} + k_{j-1})^2}\frac{\partial \kappa_{j-1}}{\partial b_1}, \quad \text{για } j = 1,4 \tag{32}$$

$$\frac{\partial \kappa_j}{\partial b_2} = -\, const\frac{b_1}{b_2^2}\frac{1}{(\delta_{i-1} + k_{j-1}/2)} - const\frac{b_1}{b_2}\frac{1}{(\delta_{i-1} + k_{j-1}/2)^2}\frac{1}{2}\frac{\partial \kappa_{j-1}}{\partial b_1}, \quad \text{για } j = 2,3 \tag{33}$$

$$\frac{\partial \kappa_j}{\partial b_2} = -\, const\frac{b_1}{b_2^2}\frac{1}{(\delta_{i-1} + k_{j-1})} - const\frac{b_1}{b_2}\frac{1}{(\delta_{i-1} + k_{j-1})^2}\frac{\partial \kappa_{j-1}}{\partial b_2}, \quad \text{για } j = 1,4 \tag{34}$$

Συνεπώς, χρησιμοποιώντας τις σχέσεις 31, 32,33, 34 και τις σχέσεις 29, 30 προκύπτει το μητρώο $\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{b}}$. Τελικά, χρησιμοποιείται η σχέση 28 και υπολογίζονται οι παράγωγοι ευαισθησίας. Τα αποτελέσματα θα παρατεθούν μαζί με τις άλλες μεθόδους για απευθείας σύγκριση.

2.2.4 Πεπερασμένες Διαφορές

Χρησιμοποιήθηκε το σχήμα των προς πίσω διαφορών (backwards FD). Η σύγκριση του βήματος φαίνεται στον πίνακα 1. Από τον πίνακα φαίνεται πως το βήμα $e = 1e - 7$ δίνει ακρίβεια 3 δεκαδικού ψηφίου που είναι και η περισσότερη που επιτυγχάνεται. Ύστερα, λαμβάνουν χώρα σφάλματα στρογγυλοποίησης.

e	1ε-7	1ε-8	1ε-9	1ε-10	1ε-11	1ε-12
$\frac{\delta F}{\delta b_1}$	-28.37220790	-28.37222154	-28.37327883	-28.39101398	-28.38760337	-29.55857780
$\frac{\delta F}{\delta b_2}$	141.86091789	141.86058479	141.85741292	141.83910934	141.82433005	138.92531569

Πίνακας 1: Σύγκριση Βήματος Πεπερασμένων διαφορών

2.2.5 Σύγκριση Μεθόδων

Οι παράγωγοι που προκύπτουν από τις 3 μεθόδους παρουσιάζονται στον πίνακα 2. Παρατηρείται πως όλες οι μεθόδοι είναι ίσες μέχρι το έκτο δεκαδικό ψηφίο, εκτός των πεπερασμένων διαφορών, που τους μοιάζουν μέχρι το τρίτο. Ενδιαφέρον παρουσιάζει πως η ευθεία διαφόριση είναι ίδια με την διακριτή συζυγή μέθοδο μέχρι το έβδομο ψηφίο.

2.3 Τμήμα 2.1.γ - Κύκλοι Βελτιστοποίησης

Έχοντας τις παραγώγους, μπορεί να υλοποιηθεί ένας κύκλος βελτιστοποίησης με τη μέθοδο της απότομης καθόδου σύμφωνα με τον τύπο:

$$\boldsymbol{b}^{new} = \boldsymbol{b}^{old} - \eta\frac{dF}{d\boldsymbol{b}} \tag{35}$$

	$\frac{\delta F}{\delta b_1}$	$\frac{\delta F}{\delta b_2}$
Finite Differences	-28.3722147287	141.8609281245
Direct Differentiation	-28.3721931234	141.8609656170
Continuous Adjoint	-28.3721930954	141.8609654770
Discrete Adjoint	-28.3721931341	141.8609656707

Πίνακας 2: Σύγκριση Παραγώγων μεταξύ των μεθόδων

Η παραπάνω διαδικασία συνεχίζεται μέχρι να ικανοποιηθεί το παραπάνω κριτήριο σύγκλισης:

$$\frac{\left|F^{new} - F^{old}\right|}{F^{old}} < e_{abs} \tag{36}$$

Επιλέχθηκε $e_{abs} = 1e - 5$. Επιπλέον, ο συντελεστής η δίνεται από τον τύπο (δηλαδή κανονικοποιείται):

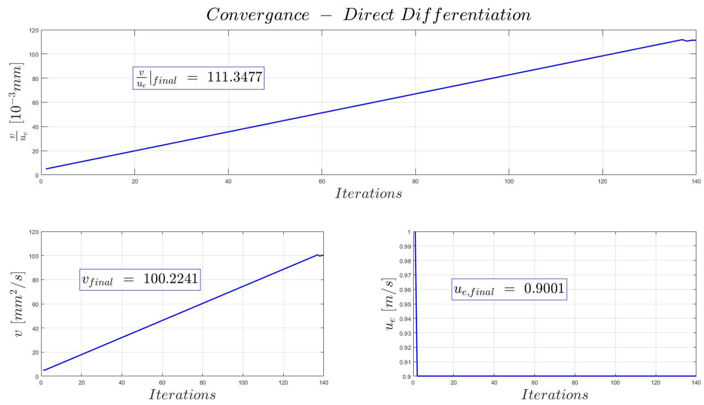
$$\eta = \frac{\eta_{desirable}}{\| \frac{dF}{d\boldsymbol{b}}\big|_{i=1} \|} \tag{37}$$

Ωστόσο, καθώς οι μεταβλητές σχεδιασμού πρέπει να είναι καθαρά θετικές, δημιουργείται ένας έλεγχος για την θετικότητα. Ακολουθείται η ακόλουθη διαδικασία:

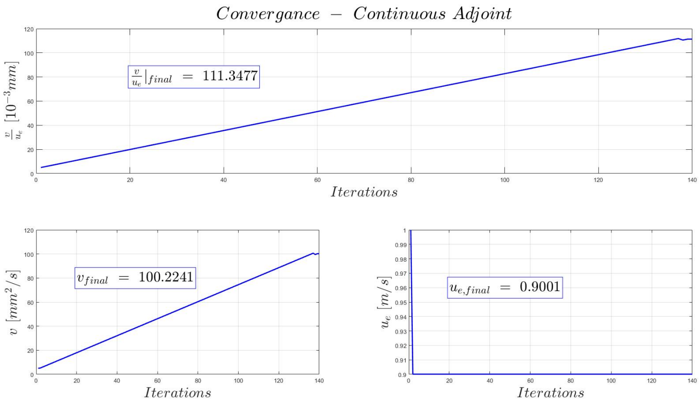
- Έλεγχος αν η μεταβολή θα πάει κάποια μεταβλητή σε αρνητική τιμή. Αν όχι, εφαρμόζεται η σχέση 35.
- Αν ναι, υπολογίζονται το $\eta_{allowed,i}$ που οδηγεί κάθε μεταβλητή στο μηδέν (Δηλαδή σε $0 < \varepsilon <<$). Σαν η για αυτό τον κύκλο βελτιστοποίησης επιλέγεται το μικρότερο από τα 2 $\eta_{allowed,i}$ πολλαπλασιασμένο με μια τιμή ώστε να μην γίνει η μεταβλητή σχεδιασμού πολύ μικρή.
- Αν σε επόμενο κύκλο η ίδια μεταβλητή ξανά τείνει να γίνει αρνητική, η μεταβλητή αυτή παγώνεται και υπολογίζεται ένα νέο η με βάση τον τύπο 36. **Τότε η μέθοδος παύει να είναι steepest descent !**

Παρακάτω, ακολουθούν τα σχήματα 2,3,4 με την σύγκλιση της βελτιστοποίησης με καθεμία από τις τρεις μεθόδους. **Παρατηρείται πως τα αποτελέσματα είναι πρακτικά ακριβώς τα ίδια.** Αυτό είναι αναμενόμενο, καθώς οι παράγωγοι διαφέρουν ελάχιστα μεταξύ τους. Σαν αρχικοποίηση λήφθηκαν οι τιμές:

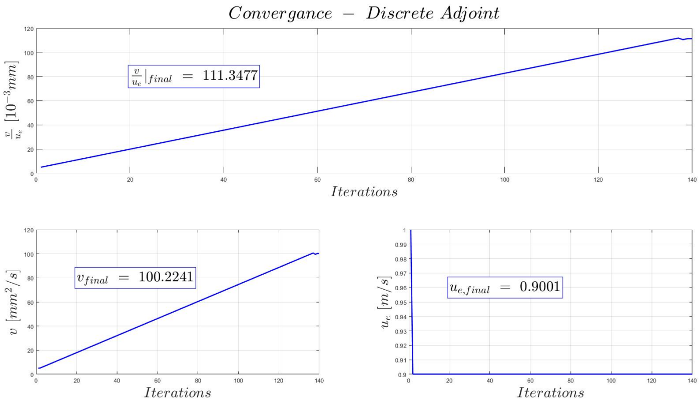
$$\{v, u_e\}^T = \{5\ [mm^2/s],\ 1\ [m/s]\}^T$$



Σχήμα 2: Σύγκλιση Βελτιστοποίησης με την μέθοδο ευθείας διαφόρισης



Σχήμα 3: Σύγκλιση Βελτιστοποίησης με την συνεχή συζυγή μέθοδο



Σχήμα 4: Σύγκλιση Βελτιστοποίησης με την διακριτή συζυγή μέθοδο

2.4 Τμήμα 2.1.δ - Διερεύνηση βήματος η

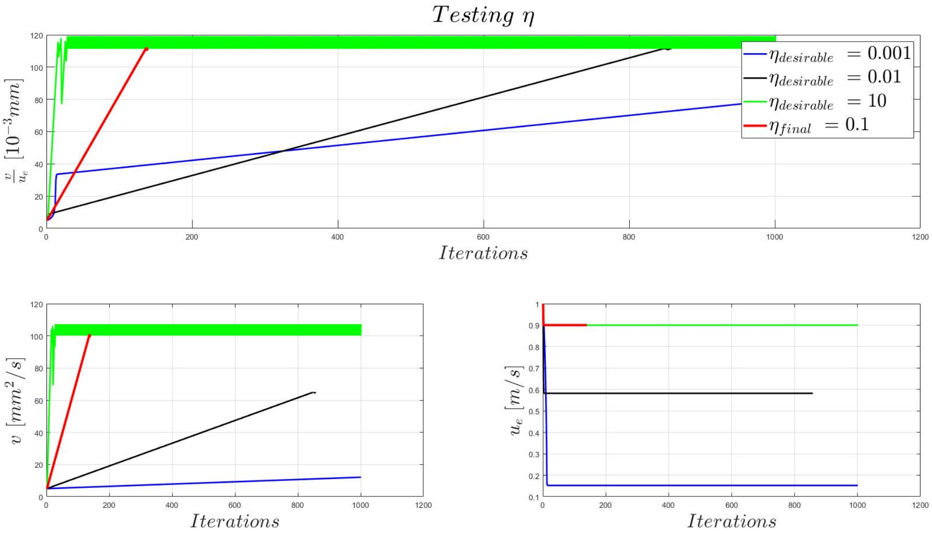
Χρησιμοποιώντας τη συνεχή συζυγή μέθοδο, θα γίνει διερεύνηση της επιλογής του βήματος η. Από τον τρόπο που λήφθηκαν οι περιορισμοί, ουσιαστικά αλλάζει η τιμή του $\eta_{desirable}$. Αφού κατά τη διάρκεια, ξαναυπολογίζεται ο συντελεστής η. Τα αποτελέσματα για διάφορες τιμές του $\eta_{desirable}$ παρουσιάζονται στο σχήμα 5.

2.5 Τμήμα 2.1.ε - Υπολογιστικό κόστος

- Οι πεπερασμένες διαφορές λύνουν $N+1$ φορές το primal ανά κύκλο βελτιστοποίησης, και συνεπώς το κόστος είναι $N+1$ TU ανά κύκλο βελτιστοποίησης. (TU time units). (Πίσω σχήμα)[3].
- Η ευθεία διαφόριση λύνει το primal και ύστερα λύνει N διαφορικές εξισώσεις ίδιου βαθμού και συνεπώς αντίστοιχου κόστους. Το τελικό κόστος είναι $N+1$ TU ανά κύκλο βελτιστοποίησης.
- Οι συζυγείς μέθοδοι λύνουν το primal και ύστερα λύνουν και μια ακόμα διαφορική (1 FAE ανά αντικειμενική συνάρτηση, αλλά εδώ είναι μια) εξίσωση αντίστοιχου βαθμού από την οποία υπολογίζονται όλες οι παράγωγοι ευαισθησίας. Συνεπώς, το κόστος είναι $2TU$ ανά κύκλο βελτιστοποίησης.

Καθώς όλες οι μέθοδοι χρειάζονται τον ίδιο αριθμό κύκλων για να συγκλίνουν, και οι μεταβλητές σχεδιασμού ήταν 2 ($N = 2$), **οι συζυγείς μέθοδοι το καταφέρνουν στα 2/3 του**

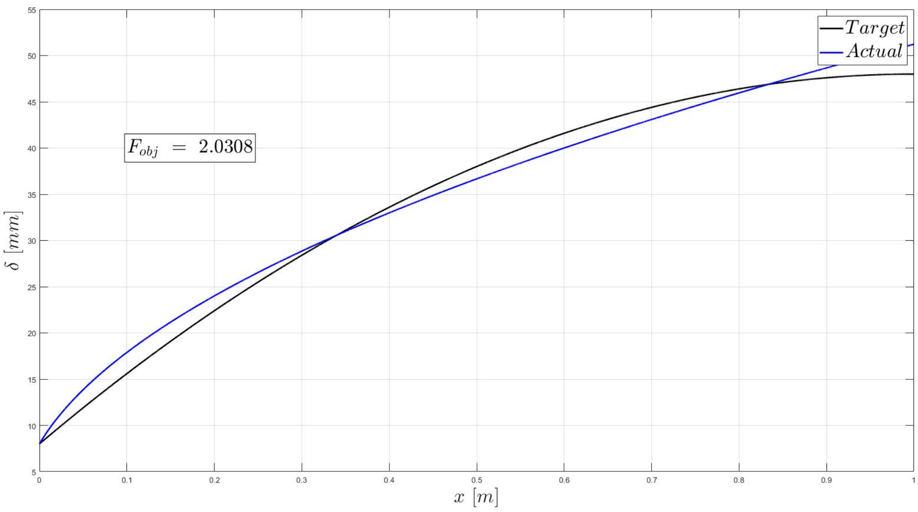
[3]Μάλιστα σε κεντρικές διαφορές το κόστος είναι $2N+1$ TU ανά κύκλο.



Σχήμα 5: Σύγκριση σύγκλισης για διάφορες τιμές του $\eta_{desirable}$

χρόνου! Φυσικά, σε ένα πρόβλημα με παραπάνω μεταβλητές σχεδιασμού, το κέρδος των συζύγων μεθόδων θα ήταν ακόμα μεγαλύτερο! Ωστόσο, προγραμματιστικά, ήταν οι πιο απαιτητικές, και συνεπώς απαιτείται επένδυση χρόνου ώστε να υλοποιηθούν.

Τέλος, παρουσιάζεται ένα διάγραμμα που δείχνει την κατανομή πάχους που επιτεύχθηκε σε σχέση με την κατανομή-στόχο, στο σχήμα 6.



Σχήμα 6: Σύγκριση κατανομή πάχους που επιτεύχθηκε σε σχέση με την κατανομή-στόχο

3. Τμήμα 2.2

Στο δεύτερο τμήμα της εργασίας ζητείται η εύρεση παραγώγων στη σχέση:

$$n_{s-s,C} = \Phi\left[\frac{r^* - \Phi\varepsilon_R}{\Phi + \varepsilon_R r^*} + \frac{1 - r^* - \Phi\varepsilon_s}{\Phi + \varepsilon_s(1 - r^*)}\right] \tag{38}$$

όπου $n_{s-s,C}$ είναι ο ισεντροπικός βαθμός απόδοσης, Φ ο συντελεστής παροχής, r^* ο θεωρητικός βαθμός αντίδρασης και $\varepsilon_R, \varepsilon_S$ είναι ο λόγος οπισθέλκουσας προς άνωση στα κινούμενα και σταθερά πτερύγια αντίστοιχα. Κάνοντας την παραδοχή $\varepsilon_R = \varepsilon_S = \varepsilon$, η σχέση έχει 3 μεταβλητές σχεδιασμού, τις Φ, ε, r^*. Συνεπώς, οι ζητούμενες παράγωγοι είναι το gradient του $n_{s-s,C}$.

$$\nabla n_{s-s,C} = \{\frac{dn_{s-s,C}}{d\Phi}, \frac{dn_{s-s,C}}{d\varepsilon}, \frac{dn_{s-s,C}}{dr^*}\} \tag{39}$$

Αυτό ζητείται να γίνει τόσο με τη μέθοδο των μιγαδικών μεταβλητών αλλά και της αυτόματης διαφόρισης. Καθώς τα αποτελέσματα ζητείται να συγκριθούν με πεπερασμένες διαφορές, αρχικά παρουσιάζεται μια σύντομη διερεύνηση του βήματος των πεπερασμένων διαφορών. Η αρχικοποίηση λήφθηκε ως:

$$\{\Phi_o, \varepsilon_o, r_o^*\} = \{0.5, 1, 0.6\}$$

η οποία στηρίζεται στο σκεπτικό πως η οπισθέλκουσα είναι ίδια με την άνωση, ενώ ο θεωρητικός συντελεστής είναι κοντά στον θεωρητικά βέλτιστο που προκύπτει όταν $\varepsilon_R = \varepsilon_S = \varepsilon$. Για τον συντελεστή παροχής λαμβάνεται τυχαία 0.5, με την υπόθεση πως η περιφερειακή ταχύτητα είναι μεγαλύτερη από την αξονική. Ύστερα, παρουσιάζεται ένας πίνακας με τις υπολογισμένες παραγώγους στο αρχικοποιημένο σημείο, για διάφορα βήματα ε.

e	$1e-8$	$1e-9$	$1e-10$	$1e-11$	$1e-12$
$\frac{dn_{s-s,C}}{d\Phi}$	-1.0098969641	-1.0098969213	-1.0098970271	-1.0098970271	-1.0098744757
$\frac{dn_{s-s,C}}{d\varepsilon}$	-0.5051525299	-0.5051525760	-0.5051525691	-0.5051525170	-0.5051965790
$\frac{dn_{s-s,C}}{dr^*}$	-0.2040608217	-0.2040608047	-0.2040608307	-0.2040607266	-0.2040555225

Πίνακας 3: Σύγκριση βήματος Πεπερασμένων Διαφορών

Από τον πίνακα 3 παρατηρείται πως για ακρίβεια 7 ψηφίου αρκεί το βήμα ε να είναι ίσο με $\varepsilon = 1e-8$. Για $\varepsilon = 1e - 12$ παρατηρείται πως αλλάζουν τα 5 ψηφία που για μικρότερες ακρίβειες ήταν σταθερά. Συνεπώς, για τόσο μικρή ακρίβεια αρχίζουν να λαμβάνουν χώρα σφάλματα στρογγυλοποίησης. Επίσης, ενώ φαινομενικά βελτιώνονται κάποιες παράγωγοι αν αυξηθεί το βήμα, ποτέ ξανά δεν βελτιώνονται όλες κατά τον ίδιο βαθμό.[4] **Συνεπώς, επειδή για διαδοχική εκλέπτυνση του βήματος, ο μέγιστος κοινός αριθμός δεκαδικών ψηφίων που ταυτίζονται είναι 7, το βήμα $\varepsilon = 1e - 8$ κρίνεται επαρκές.** Ακολουθεί ο υπολογισμός των παραγώγων με τις ζητούμενες μεθόδους.

3.1 Τμήμα 2.2.α - Αλγοριθμική Διαφόριση

Αρχικά, δημιουργήθηκε ένα λογισμικό που υπολογίζει την σχέση 38 με είσοδο τις μεταβλητές σχεδιασμού. Αυτό παρουσιάζεται παρακάτω:

```
void nis(double F,double e, double r,double* n){

*n = F*( (r-F*e)/(F+r*e) + (1-r-F*e)/(F+e*(1-r) ) );

}
```

Αυτό, εισήχθηκε στο λογισμικό Tapenade , επισημαίνωντας πως οι μεταβλητές σχεδιασμού είναι οι μεταβλητές F, e, r, ενώ η έξοδος είναι το n και προέκυψε ο επόμενος κώδικας τόσο για forward όσο και για reverse διαφόριση.

[4] Για παράδειγμα, αν συγκριθούν οι παράγωγοι με βήμα $\varepsilon = 1e - 10$ και $\varepsilon = 1e - 11$, η πρώτη παράγωγος έχει 10 ίδια δεκαδικά ενώ η δεύτερη 7, αλλά η τρίτη έχει 6.

```c
/*        Generated by TAPENADE     (INRIA, Ecuador team)
    Tapenade 3.16 (develop) - 31 May 2021 11:17
*/
/*        Generated by TAPENADE     (INRIA, Ecuador team)
    Tapenade 3.16 (develop) - 31 May 2021 11:17
*/
#include <adBuffer.h>

/*
  Differentiation of nis in forward (tangent) mode:
   variations   of useful results: *n
   with respect to varying inputs: e *n r F
   RW status of diff variables: e:in n:(loc) *n:in-out r:in F:in
   Plus diff mem management of: n:in
*/
void nis_d(double F, double Fd, double e, double ed, double r, double rd,
        double *n, double *nd) {
    double temp, temp0, temp1, temp2;
    temp = F + e*(-r+1);
    temp0 = (-(F*e)-r+1)/temp;
    temp1 = (r-F*e)/(F+r*e);
    temp2 = temp1 + temp0;
    *nd = temp2*Fd + F*((rd-e*Fd-F*ed-temp1*(Fd+e*rd+r*ed))/(F+r*e)+(-(e*Fd)-F
        *ed-rd-temp0*(Fd+(1-r)*ed-e*rd))/temp);
    *n = F*temp2;
}
/*
  Differentiation of nis in reverse (adjoint) mode:
   gradient     of useful results: *n
   with respect to varying inputs: e *n r F
   RW status of diff variables: e:out n:(loc) *n:in-out r:out
                F:out
   Plus diff mem management of: n:in
*/
void nis_b(double F, double *Fb, double e, double *eb, double r, double *rb,
        double *n, double *nb) {
    double temp, temp0, temp1, tempb, tempb0, tempb1, tempb2;
    *n = F*((r-F*e)/(F+r*e)+(1-r-F*e)/(F+e*(1-r)));
    temp = F + e*(-r+1);
    temp0 = (-(F*e)-r+1)/temp;
    temp1 = (r-F*e)/(F+r*e);
    tempb = F*(*nb)/(F+r*e);
    tempb1 = F*(*nb)/temp;
    tempb2 = -(temp0*tempb1);
    tempb0 = -(temp1*tempb);
    *Fb = (temp1+temp0)*(*nb) + tempb2 - e*tempb1 + tempb0 - e*tempb;
    *nb = 0.0;
    *eb = (1-r)*tempb2 - F*tempb1 + r*tempb0 - F*tempb;
    *rb = tempb - tempb1 - e*tempb2 + e*tempb0;
}
```

Παρατηρείται πως οι είσοδοι και οι έξοδοι του προγράμματος διπλασιάστηκαν, κάτι που **αποτελεί ένδειξη των έντονων απαιτήσεων μνήμης της μεθόδου αυτής.**

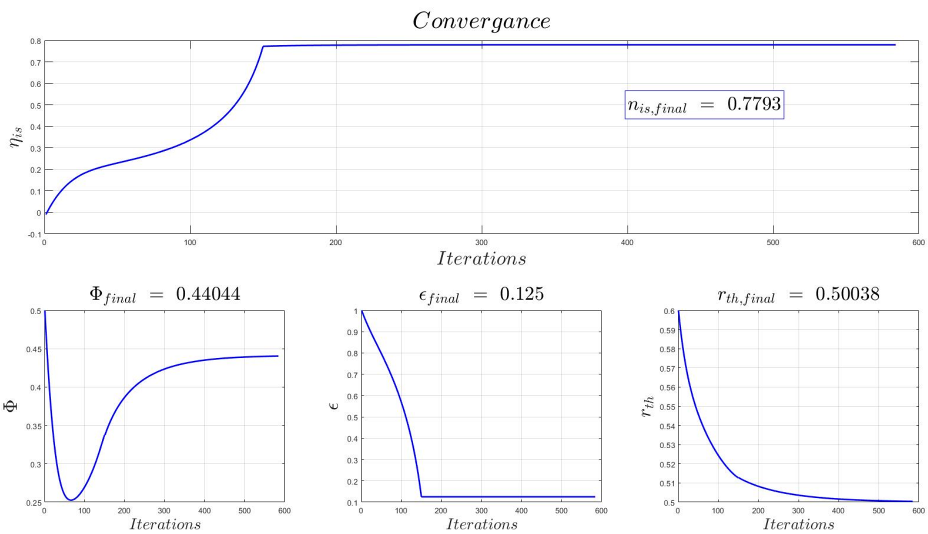
Η κύρια διαφορά των μεθόδων είναι η εξής. Η ευθεία αυτόματη διαφόριση, είναι αντίστοιχη του direct differantiation . Για να υπολογιστούν οι παράγωγοι ευαισθησίας ($\nabla \frac{F}{b_n}$), πρέπει να κληθεί 3 φορές η συνάρτηση nis_d και κάθε φορά να δίνεται σαν είσοδος σε μία από τις μεταβλητές Fd, ed rd, η μονάδα ενώ στις άλλες 2 μηδέν. (Τα υπόλοιπα arguments της συνάρτησης συμπληρώνονται κατάλληλα.) Η παάγωγος $\frac{\delta F}{\delta b_i}$ δίνεται από τη μεταβλητή nb. Όμως, η αντίστροφη αυτόματη διαφόριση είναι σαν το adjoint. Δηλαδή, καλείται μια φορά η ρουτίνα nis_b η οποία υπολογίζει το primal και ύστερα βρίσκει όλες τις παραγώγους ευαισθησίας (Στη μεταβλητή nb δίνεται μονάδα, ενώ οι έξοδοι αυτής της συνάρτησης είναι οι μεταβλητές Fb, eb, rb). Γενικά, η αντίστροφη αυτόματη διαφόριση καλείται τόσες φορές, όσο οι αντικειμενικές συναρτήσεις. **Καθώς σε αυτό το πρόβλημα υπάρχουν 3 μεταβλητές σχεδιασμού και 1 αντικειμενική, συμφέρει περισσότερο η αντίστροφη αυτόματη διαφόριση!**

Τα αποτελέσματα παρουσιάζονται στον πίνακα 4. Παρατηρείται πως η ακρίβεια τον δυο μεθόδων αυτόματης διαφόριση είναι ίδια μέχρι το δέκατο δεκαδικό ψηφίο. **Ωστόσο, στην αντίστροφη διαφόριση επιτεύχθηκε με το ένα τρίτο του κόστους!**

	$\frac{dn_{s-s,C}}{d\Phi}$	$\frac{dn_{s-s,C}}{d\varepsilon}$	$\frac{dn_{s-s,C}}{dr^*}$
FD	-1.0098969641	-0.5051525299	-0.2040608217
Algorithmic - Forward	-1.0098969493	-0.5051525355	-0.2040608101
Algorithmic - Backward	-1.0098969493	-0.5051525355	-0.2040608101

Πίνακας 4: Σύγκριση παραγώγων Αλγοριθμικής Διαφόρισης (forward και backwards mode) με πεπερασμένες διαφορές.

Χρησιμοποιώντας τη μέθοδο της απότομης ανόδου (καθώς μιλάμε για πρόβλημα μεγιστοποίησης), και φτιάχνοντας και έναν τρόπο να διαχειρίζεται τους περιορισμούς θετικότητας των μεταβλητών, έγινε η βελτιστοποίηση του συντελεστή απόδοσης και παρουσιάζεται η σύγκλιση στο σχήμα 7.



Σχήμα 7: Βελτιστοποίηση βαθμού απόδοσης πτερυγώσεων για $e_{min} = 0.125$

Παρατηρείται πως η τελική τιμή του βαθμού απόδοσης είναι ίδια με την θεωρητική τιμή που προκύπτει από τον τύπο (το $e_{min} = 0.125$ επιλέχθηκε ώστε να μην τεινει αυτό στο μηδέν και ο συντελεστής $n_{is} \to 1$) :

$$n_{is,max} = 1 + 2e^2 + 2e\sqrt{1+e^2} \tag{40}$$

3.2 Τμήμα 2.2.β - Μιγαδική Διαφόριση

Από το Taylor ανάπτυγμα:

$$F(\boldsymbol{b}+d\boldsymbol{b}) = F + \frac{dF}{d\boldsymbol{b}}d\boldsymbol{b} + O(d\boldsymbol{b}^2)$$

$$F(\boldsymbol{b}+id\boldsymbol{b}) = F + i\frac{dF}{d\boldsymbol{b}}d\boldsymbol{b} + O(d\boldsymbol{b}^2)$$

Τελικά:

$$\frac{dF}{db_i} = \lim_{e\to 0} imag\left\{\frac{F(b_i+ie)}{e}\right\} \tag{41}$$

Για την υλοποίηση του παραπάνω, δημιουργήθηκε μια συνάρτηση με μιγαδικά ορίσματα που υπολογίζει τον ισεντροπικό βαθμό απόδοσης από τη σχέση 38. Ύστερα, δοκιμάσθηκαν διάφορα βήματα του βήματος ε και τα αποτελέσματα παρουσιάζονται παρακάτω:

e	1ε-4	1ε-5	1ε-6	FD - 1ε-8
$\frac{dn_{s-s,C}}{d\Phi}$	-1.0098969494	-1.0098969493	-1.0098969493	-1.0098969641
$\frac{dn_{s-s,C}}{de}$	-0.5051525354	-0.5051525355	-0.5051525355	-0.5051525299
$\frac{dn_{s-s,C}}{dr^*}$	-0.2040608101	-0.2040608101	-0.2040608101	-0.2040608217

Πίνακας 5: Σύγκριση παραγώγων με μιγαδικές μεταβλητές ανάλογα την τιμή του βήματος ε

Παρατηρείται πως ήδη με βήμα $1e-5$ υπάρχει ακρίβεια 10 δεκαδικού ψηφίου, όταν με τις πεπερασμένες διαφορές η μέγιστη ακρίβεια που επιτεύχθηκε είναι στο 7 δεκαδικό ψηφίο. **Το σημαντικότερο όμως είναι, ότι δεν υπάρχουν σφάλματα στρογγυλοποίησης, καθώς δεν γίνεται αφαίρεση στον αριθμητή!**

4. Παράρτημα - Προγράμματα

Ακολουθούν οι κώδικες για πληρότητα. Ο κώδικας για το πρώτο μέρος είναι:

```
%%%%% IMPORTANT NOTES %%%%%
%    Step of ODE dx must divide [0,1] into n spaces, so they would be
    n+1
%    nodes. This is important for Simpson1_3, for the solution of the
    ODEs
%    that come with Direct Differantiation. There, because d = d(x),
    and we
%    do not use the analytical solution, there are 2 ways to do runge
    and
%    kutta. Either a polynomial interpolation, so we get d in between
%    nodes, or the solution of DD's ODEs has half the nodes. -->

%% Input
clc
clear all

%choosing method
method = 2; % 1 -DD, 2 - CA, 3 - DA

%Data
K1 = 6;
ue =1;                            %[m/s]
v = 1.2 ;                         %[mm^2/s]
v_ue = 1.2;                       %  *(1e-6* 10^6);  in [mm^2/s];

%boundary cond
xl = 0;
xu = 1;
do = (2+K1); %[mm]

%% ODE

dx= 0.0125/2/100;

d_x = @(x,d) pi^2*v/ue/( (4-pi)* d) ;
[d,x] = RKutta4(d_x,dx,xl,xu,do);

%target
ds = do*(1+10*x-5*x.^2) ;

%% optimization
%Initialization
b1(1) = 5;                 %  *(1e-6* 10^6);  in [mm^2/s];
b2(1) = 1 ;
Fold  = 100000;

%FD -epsilon
e = 1e-7;

%relax factors
ia =1; iaa=1;
ib = 1; ibb=1;
constraints_activeA = 0;
constraints_activeB = 0;
```

```
Number = 1000;
for  i=1:Number

d_x = @(x,d) pi^2*(b1(i)/b2(i)) /( (4-pi)* d) ;
d = RKutta4(d_x,dx,xl,xu,do);

%Convergance Plot -
% uncomment code for an animation of convergance
%----------------------------------------------
% [d,x] = RKutta4(d_x,dx,xl,xu,do);
% dana = @(x) sqrt( 2*pi^2*b1(i)/b2(i)/(4-pi) * x +do^2); %
% d_an  = dana(x);
% plot(x,d,x,ds,x,d_an)
% pause(0.1)

F = primal(b1(i),b2(i),dx,xl,xu,do,ds);
%% Termination Criterion
if abs(F-Fold)/Fold < 1e-5
     break
end
Fold = F;

%% choosing method
switch method

    %%------ Direct Differentiation
    %------------------------------------------------
    case 1
        % linear odes
        ODE1 = @(x,dh,ddb1) -pi^2*b1(i) / ( b2(i)*(4-pi)*(dh).^2) *
            ddb1 + pi^2/( b2(i) *(4-pi).*dh  );
        ODE2 = @(x,dh,ddb2) -pi^2*b1(i) / ( b2(i)*(4-pi)*(dh).^2) *
            ddb2 - pi^2*b1(i)/(b2(i)^(2) *(4-pi)*dh  );

        % solving odes (it turns dx to 2dx) -> taking derivatives
        [ddb1,xd] = RKutta4DD(ODE1,dx,x,d,0);
        [ddb2,xd] = RKutta4DD(ODE2,dx,x,d,0);

        %taking every second value
        dsd = do*(1+10*xd-5*xd.^2) ; %d_target(x)
        dd = d(1:2:end); %d(x)

        % sensitivity derivatives
        dd1 = Simpson1_3(xd,2*(dd-dsd).*ddb1 );
        dd2 = Simpson1_3(xd,2*(dd-dsd).*ddb2 );

        dF_db1 = dd1;
        dF_db2 = dd2;
        titletxt = '$Convergance\ -\ Direct\ Differentiation$';

        %%---------- Continuous Adjoint
        %------------------------------------------------
    case 2
        %direct integration of adjoint eq
        PSIother = cumtrapz(x, 2*(d-do*(1+10*x-5*x.^2))./d );
        PSIother = PSIother - PSIother(end);
```

```
        %Calculating SD
        dfdac1 = Simpson1_3(x,    -PSIother*pi^2/( (4-pi)*b2(i) )      )
            ;
        dfdac2 = Simpson1_3(x,    +PSIother*pi^2*b1(i)/( (4-pi)*b2(i)
            ^2 )      );

        % sensitivity derivatives
        dF_db1 = dfdac1;
        dF_db2 = dfdac2;
        titletxt = '$Convergance \ - \ Continuous\ Adjoint$';

        %%------------ Discrete Adjoint
        %-------------------------------------------------------
    case 3
        [A1,A3] = primalADJ(b1(i),b2(i),dx,xl,xu,do,ds);
        % sensitivity derivatives
        dF_db1 = A3(1);
        dF_db2 = A3(2);
        titletxt = '$Convergance \ - \ Discrete\ Adjoint$';

        %%------------ Finite Differences
        %-------------------------------------------------------
    otherwise
        Fe1 = primal(b1(i)+e,b2(i),dx,xl,xu,do,ds);
        Fe2 = primal(b1(i),b2(i)+e,dx,xl,xu,do,ds);


        dF_db1FD = (Fe1-F)/e ;
        dF_db2FD = (Fe2-F)/e ;

        % sensitivity derivatives
        dF_db1 = dF_db1FD;
        dF_db2 = dF_db2FD;
end


%% Correction - Constraints - 1
ndesirable = 10;
if i ==1
    eta = ndesirable /sqrt(dF_db1^2+dF_db1^2);
end

limit = 1e-3;
AA = b1(i) - eta*dF_db1;
BB = b2(i) - eta*dF_db2;
if (AA<0 | BB<0)
    AA = abs( (b1(i)-limit)/dF_db1 );
    BB = abs( (b2(i)-limit)/dF_db2 );
    if (AA<BB)
        if constraints_activeA ==0
            i
            eta = AA ;
            ia = 1/10; %relaxation
            constraints_activeA = 1;
        else
            ia = 0;
```

```
                eta = ndesirable /sqrt(dF_db1^2+dF_db1^2);
            end
        else
            if constraints_activeB ==0
                i
                eta = BB ;
                ib =1/10; %relaxation
                constraints_activeB = 1;
            else
                ib = 0;
                eta = ndesirable /sqrt(dF_db1^2+dF_db1^2);
            end


    end
end

%% new b

b1(i+1) = b1(i) - ia*eta/(iaa)*dF_db1;
b2(i+1) = b2(i) - ib*eta/(ibb)*dF_db2;


end

%% convergance plots
d_x   = @(x,d) pi^2*b1(end-1)/b2(end-1)/( (4-pi)* d) ;
[d,x] = RKutta4(d_x,dx,xl,xu,do);
dana = @(x) sqrt( 2*pi^2*b1(end-1)/b2(end-1)/(4-pi) * x +do^2); %
d_an  = dana(x);
plot(x,d,x,ds,x,d_an)

figure(1)
subplot(2,2,3)
plot(1:length(b1),b1(1:end),'LineWidth',2,'Color',[0 0 1])
%title('$Convergance$','Interpreter','latex','FontSize',30)
xlabel('$Iterations$','Interpreter','latex','FontSize',25)
ylabel('$v \ [mm^2/s]$','Interpreter','latex','FontSize',25)
text( 20,80,['$v_{final} \ = \ ',num2str( b1(end) ),'$'],'Interpreter','latex','FontSize',25,'BackgroundColor','w','EdgeColor','b')
grid on
subplot(2,2,4)
plot(1:length(b1),b2(1:end),'LineWidth',2,'Color',[0 0 1])
xlabel('$Iterations$','Interpreter','latex','FontSize',25)
ylabel('$u_e \ [m/s]$','Interpreter','latex','FontSize',25)
text( 20,0.96,['$u_{e,final} \ = \ ',num2str( b2(end) ),'$'],'Interpreter','latex','FontSize',25,'BackgroundColor','w','EdgeColor','b')
grid on
subplot(2,2,1:2)
plot(1:length(b1),b1(1:end)./b2(1:end),'LineWidth',2,'Color',[0 0 1])
title(titletxt,'Interpreter','latex','FontSize',30)
xlabel('$Iterations$','Interpreter','latex','FontSize',25)
ylabel('$\frac{v}{u_e} \ [10^{-3}mm]$','Interpreter','latex','FontSize',25)
text( 20,80,['$\frac{v}{u_e}|_{final} \ = \ ',num2str( b1(end)./b2(end) ),'$'],'Interpreter','latex','FontSize',25,'BackgroundColor','w',
```

```
    'EdgeColor','b')
grid on

figure(2)
[d,x] = RKutta4(d_x,dx,xl,xu,do);
dana = @(x) sqrt( 2*pi^2*b1(end)/b2(end)/(4-pi) * x +do^2); %
d_an  = dana(x);
plot(x,ds,'LineWidth',2,'Color',[0 0 0])
hold on
plot(x,d,'LineWidth',2,'Color',[0 0 1])
hold off
text(0.1,40,['$F_{obj} \ = \ ',num2str( F ),'$'],'Interpreter','latex'
    ,'FontSize',25,'BackgroundColor','w','EdgeColor','k')
legend('$Target$','$Actual$','Interpreter','latex','FontSize',25)
xlabel('$x \ [m]$','Interpreter','latex','FontSize',25)
ylabel('$\delta \ [mm]$','Interpreter','latex','FontSize',25)
grid on
```

Ο κώδικας για τους υπολογισμούς και τους κύκλους βελτιστοποίησης με την αυτόματη διαφόριση ακολουθεί παρακάτω.

```
#include <stdio.h>
#include <stdlib.h>

// Main function
void nisf(double F,double e, double r,double* n){

*n = F*( (r-F*e)/(F+r*e) + (1-r-F*e)/(F+e*(1-r) ) );

}

// Forward Automatic Differentiation
void nis_d(double F, double Fd, double e, double ed, double r,
    double rd,
         double *n, double *nd) {
     double temp;
     double temp0;
     double temp1;
     double temp2;
     temp = F + e*(-r+1);
     temp0 = (-(F*e)-r+1)/temp;
     temp1 = (r-F*e)/(F+r*e);
     temp2 = temp1 + temp0;
     *nd = temp2*Fd + F*((rd-e*Fd-F*ed-temp1*(Fd+e*rd+r*ed))/(F+r*e)
        +(-(e*Fd)-F
         *ed-rd-temp0*(Fd+(1-r)*ed-e*rd))/temp);
     *n = F*temp2;
}

// Reverse Automatic Differantiation
void nis_b(double F, double *Fb, double e, double *eb, double r,
    double *rb,
         double *n, double *nb) {
     double temp;
     double temp0;
     double temp1;
     double tempb;
     double tempb0;
```

```
    double tempb1;
    double tempb2;
    *n = F*((r-F*e)/(F+r*e)+(1-r-F*e)/(F+e*(1-r)));
    temp = F + e*(-r+1);
    temp0 = (-(F*e)-r+1)/temp;
    temp1 = (r-F*e)/(F+r*e);
    tempb = F*(*nb)/(F+r*e);
    tempb1 = F*(*nb)/temp;
    tempb2 = -(temp0*tempb1);
    tempb0 = -(temp1*tempb);
    *Fb = (temp1+temp0)*(*nb) + tempb2 - e*tempb1 + tempb0 - e*
        tempb;
    // *nb = 0.0;
    *eb = (1-r)*tempb2 - F*tempb1 + r*tempb0 - F*tempb;
    *rb = tempb - tempb1 - e*tempb2 + e*tempb0;
}

// Absolute Value
double absD(double a){
    if (a > 0){
        return a;
    } else {
        return -a;
    }
}


int main(){
// Initialization
double F=0.5,e= 1,r=0.6;

// For Automatic - Forward
double nis,nisd1,nisd2,nisd3;
nisf(F,e,r,&nis);

// For Automatic - Reverse
double nisR,nisdR1,nisdR2,nisdR3;
double reverse_input = 1;

// for FD
double epsilon = 1e-8;
double nise1,nise2,nise3 ;
double nised1,nised2,nised3 ;

// for optimization
double eta =0.01;
double limit = 1e-8;
double Fo,eo,ro,Obj0;
Obj0  = 150;
// for constraints
double A=100,B=100,C=100;
int constraintsActive = 0;
double var1=1,var2=1,var3= 1,eta_old = 0;
// for records
double *Fdata   = (double*)malloc(1000*sizeof(double))  ;
double *edata   = (double*)malloc(1000*sizeof(double))  ;
double *rdata   = (double*)malloc(1000*sizeof(double))  ;
```

```
double *nisdata = (double*)malloc(1000*sizeof(double))  ;
for (int i = 0; i < 1000; i++){
    Fdata[i] = 0;
    edata[i]= 0;
    rdata[i]= 0;
}
int total = 0;


printf("F= ,%lf,e= %lf, r= %lf\n",F,e,r);
// Loop (max 1000 iterations)
for (int count = 0; count<1000;count++){

Fo = F; eo =e; ro = r;
printf("obj0 = %f\n",Obj0);

    // General - prints
    printf("count is %d\n",count);
    printf("F= %lf,e= %lf, r= %lf,nis = %f\n",F,e,r,nis);

// Derivatives
#pragma region

// Automatic Differatiation:
nis_d(F,1,e,0,r,0,&nis,&nisd1);
nis_d(F,0,e,1,r,0,&nis,&nisd2);
nis_d(F,0,e,0,r,1,&nis,&nisd3);

    // prints
    printf("Automatic Differentiation - Forward:\n");
    printf("dndF = %.10lf, dnde = %.10lf, dndr = %.10lf\n",nisd1,
       nisd2,nisd3);

// Automatic Differatiation - Reverse:
nis_b(F, &nisdR1, e, &nisdR2, r,&nisdR3,
        &nisR, &reverse_input);

    // prints
    printf("Automatic Differentiation - Reverse:\n");
    printf("dndF = %.10lf, dnde = %.10lf, dndr = %.10lf\n",nisdR1,
       nisdR2,nisdR3);

// Finite Differences
nisf(F+epsilon,e,r,&nise1);
nisf(F,e+epsilon,r,&nise2);
nisf(F,e,r+epsilon,&nise3);

nised1 = (nise1-nis)/epsilon;
nised2 = (nise2-nis)/epsilon;
nised3 = (nise3-nis)/epsilon;

    // prints
    printf("Finite Differences:\n");
    printf("dndF = %.10lf, dnde = %.10lf, dndr = %.10lf\n",nised1,
       nised2,nised3);

    // for checking epsilon step
```

```
    // printf("%.10lf\n %.10lf \n %.10lf\n",nised1,nised2,nised3);

#pragma endregion

// DynamicEta
if (count ==1 ){
    eta = +0.01/ absD(nisd1);
}

// Constraints
if (F + eta * nisd1 < 0 | e + eta * nisd2 < 0.125 | r + eta * nisd3
    < 0){
A=100,B=100,C=100;
if (F + eta * nisd1 < 0){
    A = (2*limit-F)/nisd1 ;
}
if (e + eta * nisd2 < 0.125){
    B = (2*limit+0.125-e)/nisd2;
}
if (r + eta * nisd3 < 0){
    C = (2*limit-r)/nisd3;
}
// select abs(min)
    if (!constraintsActive) {
        eta_old = eta;
        constraintsActive = 1;
        if (A>B){
            if(B>C){
                eta = C;
            }else{
                eta = B;
            }
        }else {
            if(A>C){
                eta = C;
            }else{
                eta = A;
            }
        }
    }else {
        eta=eta_old;
        if (A>B){
            if(B>C){
                var3 = 0;
            }else{
                var2 = 0;;
            }
        }else {
            if(A>C){
                var3 = 0;
            }else{
                var1 = 0;
            }
        }
    }
}

```

```
nisf(F,e,r,&nis);
Obj0   = nis;

// for records
Fdata[count] = F;
edata[count] = e;
rdata[count] = r;
nisdata[count] = nis;

// New Design Variables
F = F + eta* var1 * nisd1;
e = e + eta* var2 * nisd2;
r = r + eta* var3 * nisd3;

printf("------------------------\n");
nisf(F,e,r,&nis);
// ----------------------  Terminating Condition
if ( absD(Obj0-nis) < 1e-8 ){
    printf("ENDING! count is %d\n",count);
    printf("F= %lf,e= %lf, r= %lf,nis = %f\n",F,e,r,nis);
    Fdata[count+1] = F;
    edata[count+1] = e;
    rdata[count+1] = r;
    nisdata[count+1] = nis;
    total = count+1;
    break ;
}

// end of loop
}

// Realloc for data
Fdata = (double*)realloc(Fdata,total*sizeof(double))   ;
edata = (double*)realloc(edata,total*sizeof(double))   ;
rdata = (double*)realloc(rdata,total*sizeof(double))   ;
nisdata = (double*)realloc(nisdata,total*sizeof(double))   ;

// For printing in matlab
FILE * fp = fopen("data.txt","w");


for (int i = 0; i < total; i++)
{
    fprintf(fp, "%lf, %lf, %lf, %lf\n", Fdata[i],edata[i],rdata[i],
        nisdata[i]);
}

 fclose(fp);
 free(Fdata);
 free(rdata);
 free(edata);
 free(nisdata);



    return 0;
}
```

Ο κώδικας για τους υπολογισμούς παραγώγων με τη μέθοδο των μιγαδικών μεταβλητών ακολουθεί παρακάτω:

```c
#include <stdio.h>
#include <complex.h>


// Main function
void  nisf(double complex F,double complex e, double complex r,
    double complex * n){

*n = F*( (r-F*e)/(F+r*e) + (1-r-F*e)/(F+e*(1-r) ) );

}

int main(){
double complex F=0.5 + 0*I ,e= 1+0*I,r=0.6+0*I;
double complex nis,nisd1,nisd2,nisd3,nisFI,niseI,nisrI;
double epsilon = 1e-5;

nisf(F,e,r,&nis );
nisf(F+epsilon*I,e,r,&nisFI );
nisf(F,e+epsilon*I,r,&niseI );
nisf(F,e,r+epsilon*I,&nisrI );

nisd1 = nisFI/epsilon;
nisd2 = niseI/epsilon;
nisd3 = nisrI/epsilon;

// nis = nis+epsilon;


printf("n       = %.8f\n"   ,creal(nis) );
printf("nisd1   = %.10f \n",cimag(nisd1) );
printf("nisd2   = %.10f \n",cimag(nisd2) );
printf("nisd3   = %.10f \n",cimag(nisd3) );

    return 0;
}
```