ΕΡΓΑΣΙΕΣ ΔΙΑΦΟΡΙΚΩΝ ΕΞΙΣΩΣΕΩΝ

Παραδοτέες μέχρι 10/01/20

Α.Κίνηση μηχανικού συστήματος με μεταβαλλόμενη μάζα.

Πυροσβεστικό αεροπλάνο μάζας M_0=5 τόνων εκτελεί χαμηλή πτήση πάνω από την επιφάνεια της θάλασσας και τη χρονική στιγμή t=0 αγγίζει την επιφάνειά της με ταχύτητα v_0=360 km/h για να γεμίσει τη δεξαμενή του με νερό. Το άνοιγμα της καταπακτής του είναι τέτοιο ώστε η δεξαμενή να γεμίζει με ρυθμό $dm/dt=\lambda$=100 kg/s, ενώ η χωρητικότητά της είναι m_0=2 τόνοι. Η τριβή ολίσθησης του αεροπλάνου στο νερό έχει συντελεστή b=20 kg/s. Η κίνηση του αεροπλάνου κατά τη φόρτωση με νερό καθορίζεται από τη διαφορική εξίσωση πρώτης τάξης:

$$\frac{dv}{dt} = -\frac{\lambda + b}{M_0 + \lambda t} v \quad \text{με αρχική συνθήκη} \quad v(t=0) = v_0 \quad \text{και λύση} \quad v = v_0 \left(\frac{M_0}{M_0 + \lambda t} \right)^{(\lambda+b)/\lambda}.$$

Η φόρτωση γίνεται χωρίς το αεροπλάνο να αυξάνει ισχύ και ολοκληρώνεται τη στιγμή $t_0=m_0/\lambda$=20 s, οπότε η ταχύτητα φτάνει στην ελάχιστη τιμή:

$$v(t_0) = v_0 \left(\frac{M_0}{M_0 + m_0} \right)^{(\lambda+b)/\lambda} = 240 \text{ km/s}.$$

Τότε το αεροπλάνο κλείνει την καταπακτή και αυξάνει ισχύ για να επιταχύνει και να ανασηκωθεί. Επιλέγοντας κατάλληλο βήμα, λύστε τη διαφορική εξίσωση της ταχύτητας συναρτήσει του χρόνου κατά τη φόρτωση, δηλαδή στο διάστημα $[0,t_0]$, με τη βελτιωμένη μέθοδο Euler (ειδική περίπτωση Runge-Kutta δεύτερης τάξης). Σχεδιάστε την αριθμητική και την αναλυτική λύση στο ίδιο σχήμα, για να τις συγκρίνετε και να ελέγξετε την ακρίβειά σας.

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

double f (double t, double v, double parameters[3]);

int main (void)
{
    FILE* fp=fopen("NC_diffeq_1.txt", "w");
    /*creating the files for our results*/
    double parameters[3] = {5000, 100, 20}, m_0=2000;/*parameters*/
    double t_start=0, t_end=m_0/(parameters[1]+0.);/*interval borders*/
    int Np = 20; /*number of points*/
    double v=360, h=(t_end-t_start)/(Np+0.), F, t=0;

    printf("\n t (sec)\t v (km/s)\n %lf\t %lf", t, v);
    fprintf(fp, "%lf \t,\t %lf\n", t, v);
    for(int i=1; i<=Np; i++){
        t = t_start + h*i;
        F = f(t_start+h*i, v, parameters);
        v = v + (h/(2+0.))*(F + f(t+h, v+h*F, parameters));
        /*applying the improved Euler method*/
        fprintf(fp, "%lf \t,\t %lf\n", t, v);
        /*saving results*/
        printf("\n %lf\t %lf", t, v);
    }
    fclose(fp);
    return 0;
}

double f (double t, double v, double parameters[3])
{
    double M_0=parameters[0], L=parameters[1], b=parameters[2];
    return -((L + b)/(M_0 + L*t +0.))*v;
}
```

code block 0

Και τα αποτελέσματα είναι:

diagram for question A

Όπου με πορτοκαλί είναι η αναλυτική λύση και με μαύρο η αριθμητική. Η ακρίβεια κρίνεται ικανοποιητική

Β.Κίνηση μηχανικού συστήματος με μεταβαλλόμενη επιτάχυνση.

Ένα σώμα εκτοξεύεται κατακόρυφα από την επιφάνεια της Γης προς τα πάνω με ταχύτητα v_0=16 km/s. Παραλείποντας την τριβή στην ατμόσφαιρα, η εξίσωση της κίνησής του στο βαρυτικό πεδίο της Γης, το οποίο μεταβάλλεται καθώς το σώμα ανυψώνεται γρήγορα, είναι μια διαφορική εξίσωση πρώτης τάξης:

$$\frac{dv}{dx} = -\frac{gR^2}{v(R+x)^2} \quad \text{με αρχική συνθήκη} \quad v(x=0) = v_0 \quad \text{και λύση} \quad v = \sqrt{v_0^2 - \frac{2gRx}{R+x}}\,,$$

όπου x είναι η απόσταση του σώματος (το ύψος) από την επιφάνεια της Γης, g=9.8 m/s^2 είναι η ένταση του πεδίου βαρύτητας στην επιφάνεια της Γης και R=6375 km είναι η ακτίνα της Γης. Η ταχύτητα διαφυγής (ελάχιστη v_0=v_e για να μην επιστρέψει το σώμα στη Γη) βρίσκεται από το όριο $v \to 0$ της αναλυτικής λύσης όταν $x \to \infty$:

$$v_e = \sqrt{2gR} = 11.1 \text{ km/s}.$$

Επομένως, με την αρχική συνθήκη του προβλήματος, το σώμα θα διαφύγει. Επιλέγοντας κατάλληλο βήμα, λύστε τη διαφορική εξίσωση της ταχύτητας συναρτήσει του ύψους στο διάστημα [0,2R] με τη μέθοδο Runge-Kutta τέταρτης τάξης και σχεδιάστε την αριθμητική και την αναλυτική λύση στο ίδιο σχήμα, για να τις συγκρίνετε και να ελέγξετε την ακρίβειά σας.

Ερώτηση Β. Επιλέχθηκε η προσέγγιση να γίνει με 15 σημεία. Ο κώδικας γι αυτό το ερώτημα φαίνεται παρακάτω:

```c
double f (double x, double v, double parameters[2]);
double v_analytic (double x, double parameters[2]);

int main (void)
{
    FILE *fp = fopen("NC_diffeq_2.txt", "w");
    char* ferr="NC_diffeq_err_2.txt"; clear_file(ferr);
    /*creating the files for our results*/
    double parameters[2] = {0.00981, 6375};/*parameters*/
    double x_start=0, x_end=2*parameters[1];/*interval borders*/
    int Np = 15; /*number of points*/
    double v=16, h=(x_end-x_start)/(Np+0.), x=0, err, V;
    double k1, k2, k3, k4;

    printf("\n x (km)\t\t v (km/s) \t err %c \n %lf\t %lf", '%', x, v);
    for(int i=1; i<=Np; i++){
        x = x_start+h*i;
        k1 = f(x_start+h*(i-1), v, parameters);
        k2 = f(x_start+h*(i-1)+h*0.5, v+(h*0.5)*k1, parameters);
        k3 = f(x_start+h*(i-1)+h*0.5, v+(h*0.5)*k2, parameters);
        k4 = f(x_start+h*(i-1)+h, v+h*k3, parameters);

        v = v + (h/(6+0.))*(k1 + 2*k2 + 2*k3 + k4);
        /*applying the 4th order RK method*/
        fprintf(fp, "%lf \t,\t %lf", x, v);
        /*saving results*/
        V = v_analytic(x, parameters);
        err = ((fabs(v-V))/(V+0.))*100;
        addto(err, ferr);
        /*calculating divergence from analytical result*/
        printf("\n %lf\t %lf\t %lf", x, v, err);
    }
    fclose(fp);
    return 0;
}
```

code block 1

Με συναρτήσεις τις εξής:

```
double f (double x, double v, double parameters[2])
{
    double g=parameters[0], R=parameters[1];
    return -(g*pow(R,2))/(v*pow(R+x,2)+0.);
}

double v_analytic (double x, double parameters[2])
{
    double g=parameters[0], R=parameters[1], v_0=16;
    return sqrt(pow(v_0,2)- (2*g*R*x)/(R+x+0.));
}
```

code block 2

Και προκύπτει η εξής παράσταση:

diagram for question B

Όπου με πορτοκαλί είναι η αναλυτική λύση και με μπλε διακεκομμένες η αριθμητική. Η ακρίβεια ήταν πολύ καλή, με μέγιστο σφάλμα περίπου 0.0001 %

Γ.Ταλαντώσεις κυκλώματος RLC σε σειρά.

Η εκφόρτιση ενός κυκλώματος με ωμική αντίσταση R, αυτεπαγωγή L και χωρητικότητα C συνδεμένες σε σειρά παριγράφεται από τη διαφορική εξίσωση δεύτερης τάξης:

$$\frac{d^2Q}{dt^2} + 2\alpha\frac{dQ}{dt} + \omega_0^2 Q = 0 \quad \text{με αρχικές συνθήκες} \quad Q(t=0) = Q_0 \quad \text{και} \quad \frac{dQ}{dt}(t=0) = 0,$$

όπου t είναι ο χρόνος, Q το φορτίο του κυκλώματος και οι συντελεστές α και ω_0 είναι:

$$\alpha = \frac{R}{2L} \quad \text{και} \quad \omega_0^2 = \frac{1}{LC}\,.$$

Οι συντελεστές αυτοί μετριούνται σε Hz και το φορτίο σε Cb. Η αναλυτική λύση αυτής της διαφορικής εξίσωσης εξαρτάται από τη διαφορά $\omega_N^2 = \omega_0^2 - \alpha^2 = -\omega'^2$ και διακρίνεται σε τέσσερις περιπτώσεις:

1. $\alpha = 0 \quad (R = 0) \quad \Rightarrow \quad Q = Q_0\cos(\omega_0 t)$, μη αποσβενόμενη,
2. $\alpha^2 < \omega_0^2 \quad (R^2 < 4L/C) \quad \Rightarrow \quad Q = Q_0(\omega_0/\omega_N)e^{-\alpha t}\sin(\omega_N t + \phi)$, υπο-αποσβενόμενη,
3. $\alpha^2 = \omega_0^2 \quad (R^2 = 4L/C) \quad \Rightarrow \quad Q = Q_0(1+\alpha t)e^{-\alpha t}$, κρίσιμα αποσβενόμενη,
4. $\alpha^2 > \omega_0^2 \quad (R^2 > 4L/C) \quad \Rightarrow \quad Q = Q_0(\omega_0/\omega')e^{-\alpha t}\sinh(\omega' t + \phi')$, υπερ-αποσβενόμενη,

όπου $\phi = \tan^{-1}(\omega_N/\alpha)$ και $\phi' = \tanh^{-1}(\omega'/\alpha)$. Επιλέγοντας κατάλληλο βήμα σε κάθε περίπτωση, λύστε τη διαφορική εξίσωση με την απλή μέθοδο Euler για τις εξής περιπτώσεις:

1. $\alpha = 0$,
2. $\alpha = 30$ MHz,
3. $\alpha = 50$ Mhz,
4. $\alpha = 60$ MHz,

με ω_0=50 MHz και Q_0=10 μCb. Σχεδιάστε την αριθμητική και την αναλυτική λύση στο ίδιο σχήμα σε καθεμιά περίπτωση, για να τις συγκρίνετε και να ελέγξετε την ακρίβειά σας.

Ερώτηση Γ. Εδώ επιλέχθηκαν 100 σημεία για την προσέγγιση. Ο παρακάτω αλγόριθμος περιλαμβάνει όλες τις περιπτώσεις για το α.

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

double f(double t, double Q, double dQ, double parameters[2]);
char integer[10] = "0123456789";/*for iterating filenames*/

int main (void)
{
    FILE *fp;
    char filename[11] = "NCd_3__.txt";
    double a[4]={0, 30, 50, 60}, parameters[2]={0, 50};
    double t=0, Q, dQ;
    int Np=100;
    double t_start=0, t_end=0.4, h=(t_end-t_start)/(Np+0.);

    for (int j=0; j<4; j++){
        filename[11]='\0';
        filename[6] = integer[j+1];
        fp = fopen(filename, "w");
        parameters[0] = a[j];/*iterating for different calues of a*/
        Q=10, dQ=0;/*starting conditions*/
        for (int i=1; i<=Np; i++){
            t = t_start + h*i;
            Q = Q + h*dQ;                          /*estimating Q*/
            dQ = dQ + h*f(t, Q, dQ, parameters);/*estimating dQ/dt*/

            fprintf(fp, "%lf \t,\t %lf\n", t, Q);/*saving results*/
        }
    }
    return 0;
}

double f(double t, double Q, double dQ, double parameters[2])
{
    double a=parameters[0], o=parameters[1];
    return -2*a*dQ - pow(o,2)*Q;
}
```

code block 3

Κι εδώ θα παρατεθούν όλα τα αντίστοιχα διαγράμματα. Σε όλα η αναλυτική λύση είναι με πορτοκαλί και η αριθμητική με μαύρο.

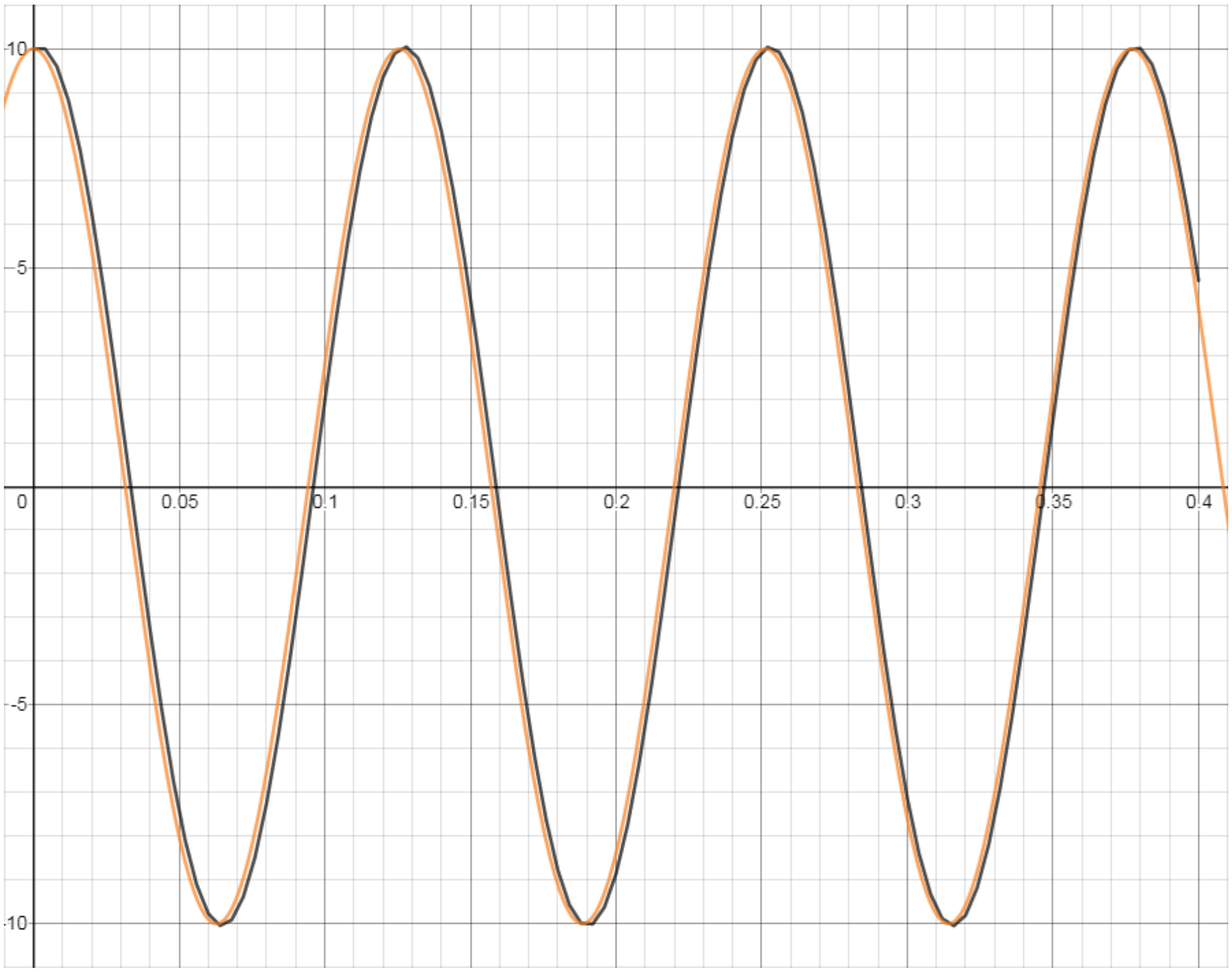
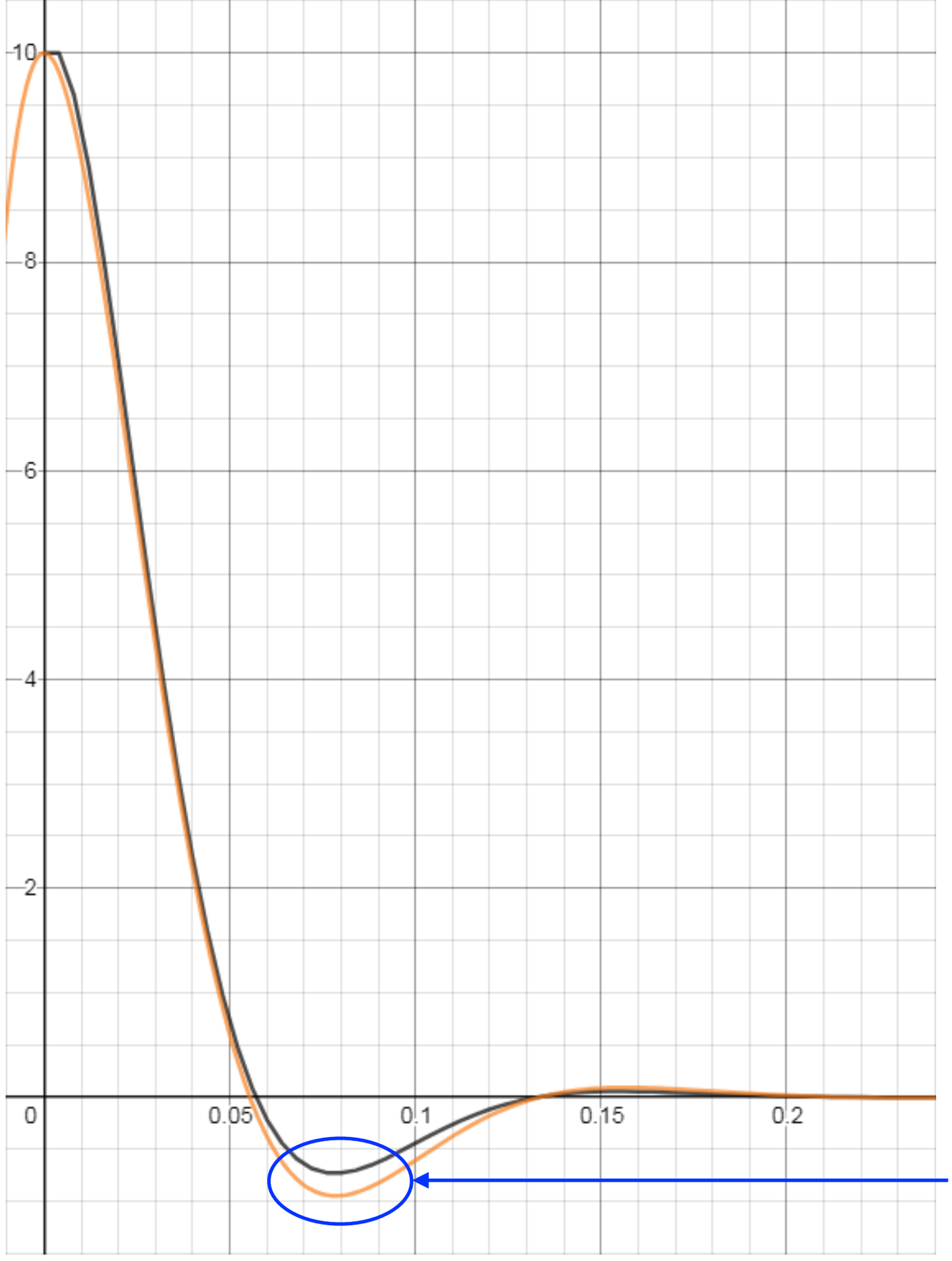


diagram for question C and a=0



Σημαντικό τοπικό σφάλμα (~25%) εκεί όπου η 2η παράγωγος της λύσης γίνεται μέγιστη.

diagram for question C and a=30

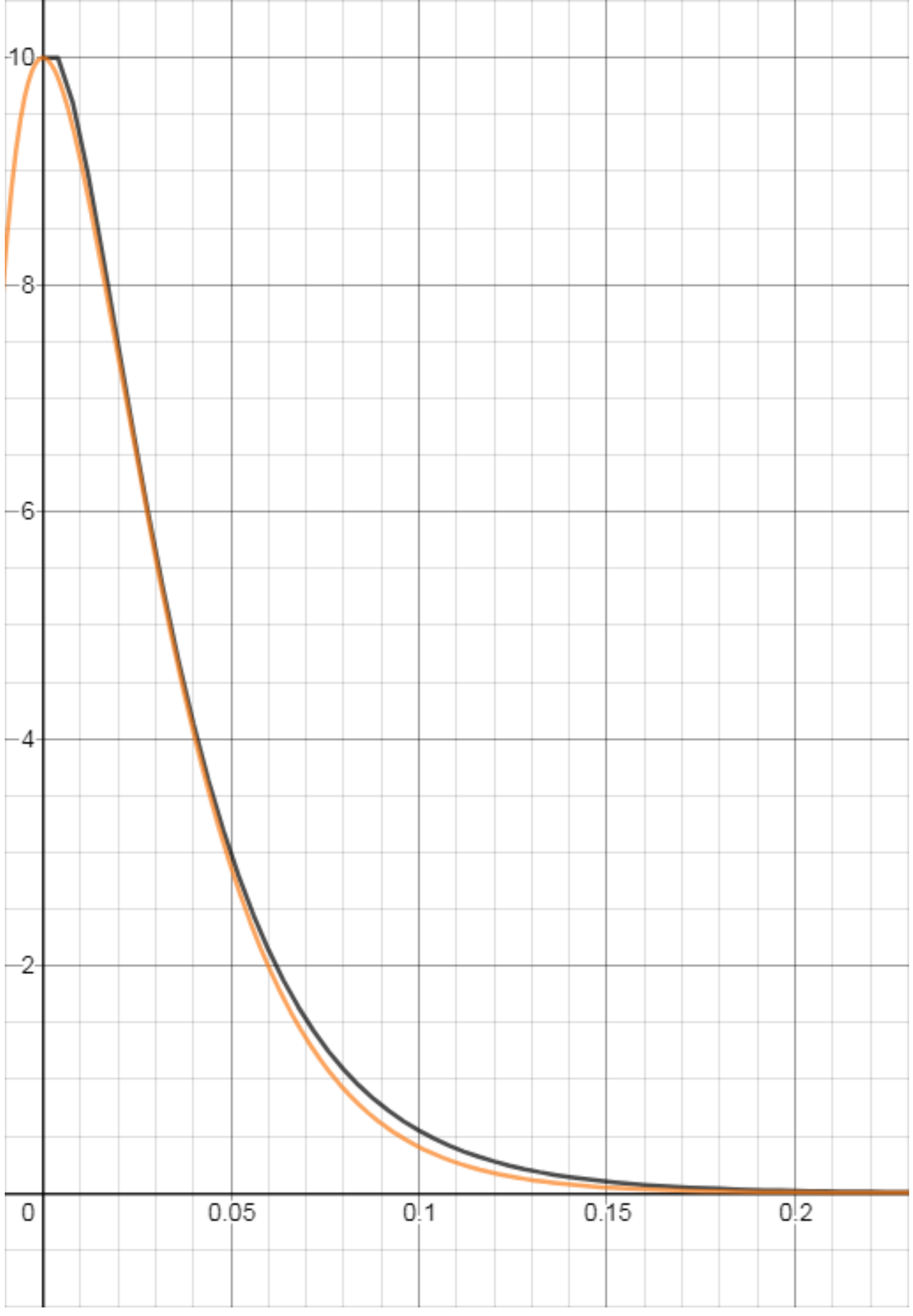


diagram for question C and a=50

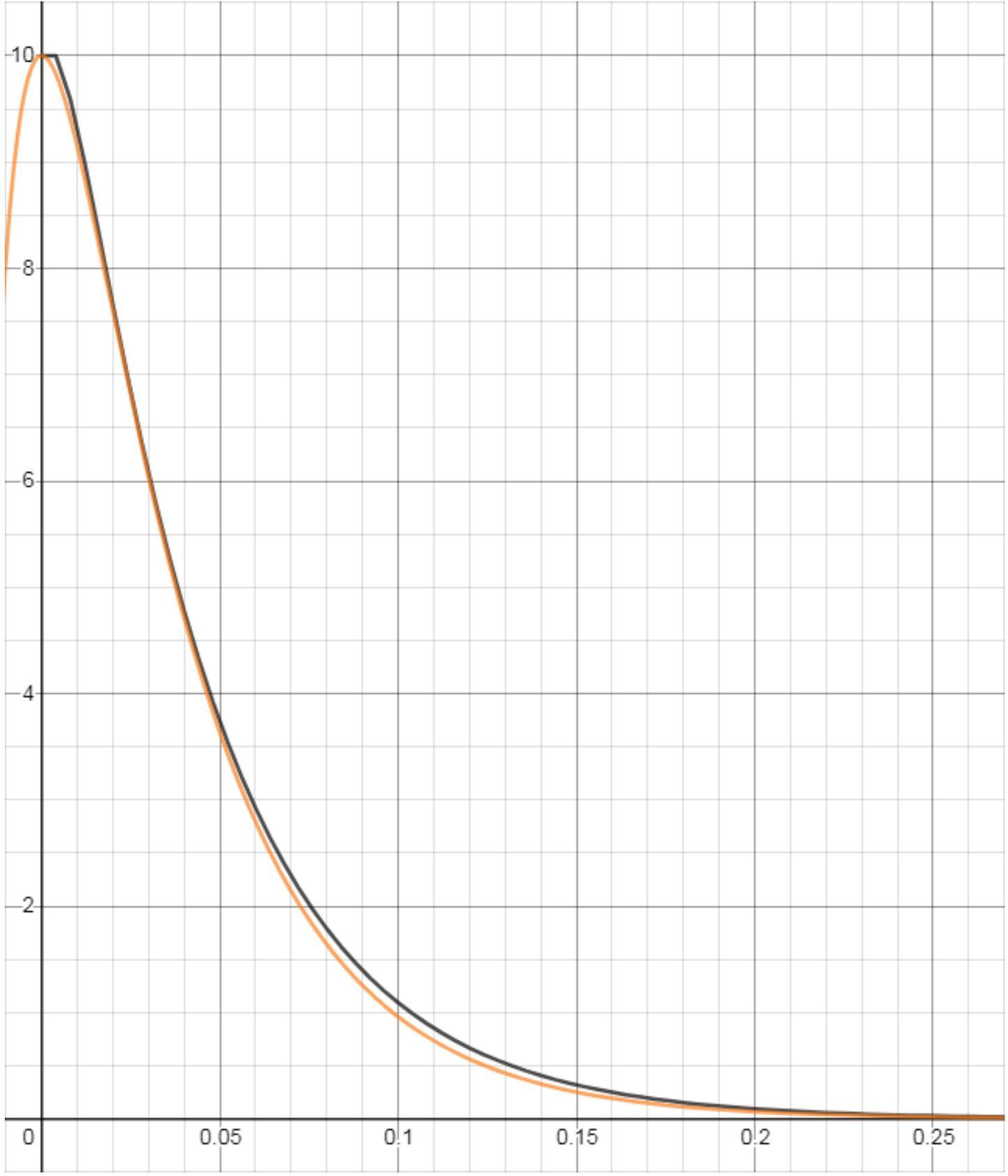


diagram for question C and a=60

Η ακρίβεια κρίνεται ικανοποιητική, εκτός ίσως από την περίπτωση α=30.

Δ.Μέθοδος πρόβλεψης-διόρθωσης Milne-Simpson.

Γράψτε τον αλγόριθμο (μόνο τους αναλυτικούς τύπους) για μια μέθοδο επίλυσης διαφορικών εξισώσεων πρώτης τάξης με πρόβλεψη βασισμένη στη μέθοδο Milne και διόρθωση βασισμένη στη μέθοδο Simpson, ακολουθώντας τα εξής βήματα:

1. Γράψτε πρώτα τον αλγόριθμο πρόβλεψης-διόρθωσης για διαφορικές εξισώσεις πρώτης τάξης.

2. Στη συνέχεια, μετατρέψτε τον αλγόριθμο ώστε να λύνει διαφορικές εξισώσεις δεύτερης τάξης (επιλύοντας ουσιαστικά ένα σύστημα δύο διαφορικών εξισώσεων πρώτης τάξης).

3. Αναλύστε τους τύπους του αλγόριθμου δεύτερης τάξης ώστε να φαίνονται οι συσχετισμοί μεταξύ των βοηθητικών μεταβλητών που χρησιμοποιούνται για την επίλυση της εξίσωσης δεύτερης τάξης.

4. Εξηγήστε πώς πρέπει να εφαρμόζεται ο αλγόριθμος: (i) Κάνοντας πρώτα πρόβλεψη Milne και για τις δύο εξισώσεις πρώτης τάξης και μετά διόρθωση Simpson για τις δύο προβλέψεις; (ii) ή κάνοντας πρόβλεψη και διόρθωση πρώτα για τη μία εξίσωση και μετά πρόβλεψη και διόρθωση για την άλλη;

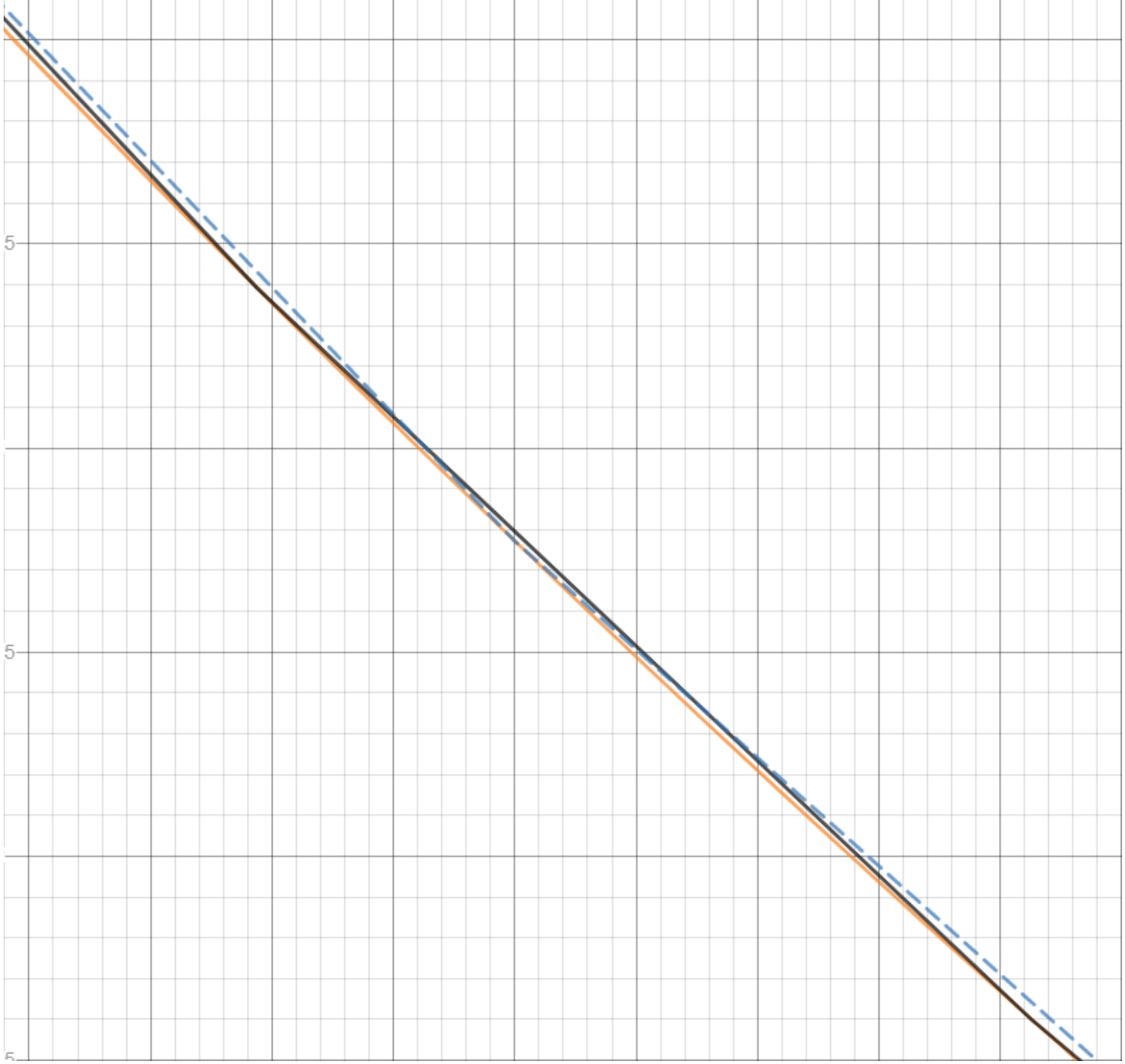
Ερώτηση Δ. Κατασκευάστηκε μια αυτοτελής συνάρτηση:

```
1  void Milne_Simpson (double (*f)(double t, double y), double t_0, double y_0,
2  double t_end, int Np, double *t, double *y)
3  /*solves differential equations using the Milne- Simpson method*/
4  {
5      double h = (t_end-t_0)/(Np+0.);/*the step*/
6      double k1, k2, k3, k4;/*this will be needed for the 4th order RK*/
7      t[0]=t_0; y[0]=y_0;/*starting points*/
8
9      for(int j=1; j<4; j++){/*applying RK to find the first 4 points*/
10         t[j] = t[0] + h*j;
11         k1 = (*f)(t[j-1], y[j-1]);
12         k2 = (*f)(t[j-1] + h*0.5, y[j-1] + (h*0.5)*k1);
13         k3 = (*f)(t[j-1] + h*0.5, y[j-1] + (h*0.5)*k2);
14         k4 = (*f)(t[j-1] + h, y[j-1] + h*k3);
15
16         y[j] = y[j-1] + (h/(6+0.))*(k1 + 2*k2 + 2*k3 + k4);
17     }
18
19     for(int i=4; i<=Np; i++){
20         t[i] = t[0] + h*i;
21
22         y[i] = y[i-4] + ((4*h)/(3+0.))*(2*(*f)(t[i-1], y[i-1]) -
23         (*f)(t[i-2], y[i-2]) + 2*(*f)(t[i-3], y[i-3]));
24         /*prediction using Milne*/
25         y[i] = y[i-2] + (h/(3+0.))*((*f)(t[i], y[i]) +
26         4*(*f)(t[i-1], y[i-1]) + (*f)(t[i-2], y[i-2]));
27         /*correction using Simpson*/
28     }
29 }
```

Ο pointer επιτρέπει τη χρήση οποιασδήποτε συνάρτησης που περνά σαν όρισμα της μεθόδου (void) Milne-Simpson.

code block 4

Όπως φαίνεται, χρησιμοποιεί την μέθοδο Runge- Kutta 4ης τάξης για να βρει τα πρώτα 3 άγνωστα σημεία και μετά χρησιμοποιεί την μέθοδο πρόβλεψης-διόρθωσης. Μια εφαρμογή στο πρόβλημα της ερώτησης Β (πάλι με 15 σημεία) μας δίνει μέγιστο σφάλμα περίπου 0.0003 %, που είναι μεγαλύτερο από αυτό της Runge- Kutta 4ης τάξης. Για να δούμε κάποια διαφορά θα πρέπει να κοιτάξουμε την παράσταση πιο κοντά:



helper diagram for question D

Το τοπικό σφάλμα αλλάζει πρόσημο από βήμα σε βήμα.

Παρατηρείται μια ¨ταλάντωση' της αριθμητικής λύσης που ίσως προκύπτει διότι ο αλγόριθμος διορθώνει συνεχώς την πορεία του.

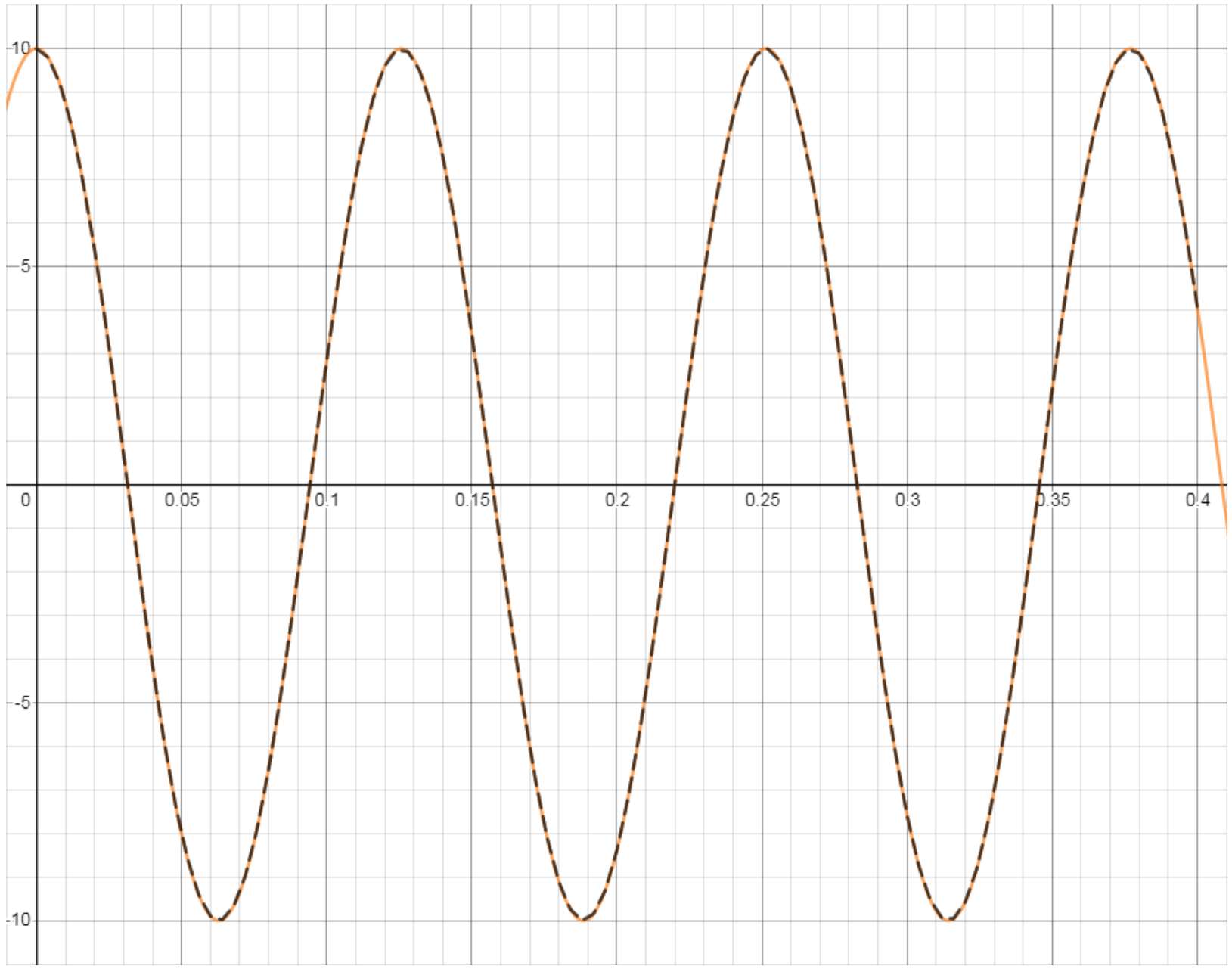
Αν θέλουμε με αυτή την μέθοδο να επιλύσουμε διαφορικές δεύτερης τάξης, θα πρέπει να τροποποιήσουμε τον αλγόριθμο:

```c
void Milne_Simpson_2 (double (*f)(double t, double y, double dy), double t_0,
double y_0, double dy_0, double t_end, int Np, double *t, double *y, double *dy)
{
    double h = (t_end-t_0)/(Np+0.), h2 = h*0.5;/*the step*/
    double k1, k2, k3, k4, kd1, kd2, kd3, kd4;
    t[0]=t_0; y[0]=y_0; dy[0]=dy_0;/*starting points*/

    for(int j=1; j<4; j++){/*applying RK to find the first 4 points*/
        t[j] = t[0] + h*j;
        kd1 = (*f)(t[j-1], y[j-1], dy[j-1]);
        kd2 = (*f)(t[j-1] + h2, y[j-1] + h2*dy[j-1], dy[j-1] + h2*kd1);
        kd3 = (*f)(t[j-1] + h2, y[j-1] + h2*(dy[j-1] + h2*kd1), dy[j-1] + h2*kd2);
        kd4 = (*f)(t[j-1] + h, y[j-1] + h*(dy[j-1] + h2*kd1), dy[j-1] + h*kd3);

        k1 = dy[j-1];
        k2 = dy[j-1] + h2*kd1;
        k3 = dy[j-1] + h2*kd2;
        k4 = dy[j-1] + h*kd3;

        y[j] = y[j-1] + (h/(6+0.))*(k1 + 2*k2 + 2*k3 + k4);
        dy[j] = dy[j-1] + (h/(6+0.))*(kd1 + 2*kd2 + 2*kd3 + kd4);
    }

    for(int i=4; i<=Np; i++){
        t[i] = t[0] + h*i;
        dy[i] = dy[i-4] + ((4*h)/(3+0.))*(2*(*f)(t[i-1], y[i-1], dy[i-1]) -
        (*f)(t[i-2], y[i-2], dy[i-2]) + 2*(*f)(t[i-3], y[i-3], dy[i-3]));
        y[i] = y[i-4] + ((4*h)/(3+0.))*(2*dy[i-1] -
        dy[i-2] + 2*dy[i-3]);
        /*predictions using Milne*/

        dy[i] = dy[i-2] + (h/(3+0.))*((*f)(t[i], y[i], dy[i]) +
        4*(*f)(t[i-1], y[i-1], dy[i-1]) + (*f)(t[i-2], y[i-2], dy[i-2]));
        y[i] = y[i-2] + (h/(3+0.))*(dy[i] +
        4*dy[i-1] + dy[i-2]);
        /*corrections using Simpson*/
    }
}
```

code block 5

Όπου προτιμήθηκε η επιλογή (i), καθώς δεν κατάφερα να το υλοποιήσω με τον δεύτερο τρόπο. Ο τύπος της διόρθωσης Simpson χρειαζόταν δεδομένα και για τη συνάρτηση και για την παράγωγό της στο σημείο που γινόταν η διόρθωση. Έτσι, έπρεπε να γίνει πρόβλεψη για να έχουμε εκτιμήσεις και για τα δύο.

Μια εφαρμογή του αλγορίθμου αυτού στο πρόβλημα της ερώτησης 3 για την πρώτη περίπτωση (α=0) και με 100 σημεία, έδωσε αυτό το διάγραμμα:



helper diagram for question D+

Από όπου παρατηρείται πως η προσέγγιση είναι καλύτερη της απλής μεθόδου Euler, και έχει ένα μέγιστο σφάλμα περίπου 0.14 %.

Για το πρόβλημα αρχικών τιμών 2ης τάξης: $y'' = f(x, y, y')$ $y(x_0) = y_0$ $y'(x_0) = y_0'$

αφού το ανάγουμε σε σύστημα 2 εξισώσεων 1ης τάξης: $y_1' = y'$ $y_2' = f(x, y, y')$

ελέγχουμε πώς μπορεί να εφαρμοστεί μέθοδος Milne (predictor) - Simpson (corrector) σε n βήματα μήκους h, με $i = 3(1)(n-1)$, λύνοντας στα πρώτα 4 σημεία, π.χ., με Runge-Kutta:

(i)

$$y_{i+1}^{(p)} = y_{i-3} + \frac{4h}{3}\left(2y'_i - y'_{i-1} + 2y'_{i-2}\right)$$

$$y'^{(p)}_{i+1} = y'_{i-3} + \frac{4h}{3}\left[2f(x_i, y_i, y'_i) - f(x_{i-1}, y_{i-1}, y'_{i-1}) + 2f(x_{i-2}, y_{i-2}, y'_{i-2})\right]$$

$$y_{i+1}^{(c)} = y_{i-1} + \frac{h}{3}\left(y'_{i+1} + 4y'_i + y'_{i-1}\right)$$

Εδώ μπαίνουν οι τιμές πρόβλεψης της y και της y'.

$$y'^{(c)}_{i+1} = y'_{i-1} + \frac{h}{3}\left[f(x_{i+1}, y_{i+1}, y'_{i+1}) + 4f(x_i, y_i, y'_i) + f(x_{i-1}, y_{i-1}, y'_{i-1})\right]$$

(ii)

$$y_{i+1}^{(p)} = y_{i-3} + \frac{4h}{3}\left(2y'_i - y'_{i-1} + 2y'_{i-2}\right)$$

$$y_{i+1}^{(c)} = y_{i-1} + \frac{h}{3}\left(y'_{i+1} + 4y'_i + y'_{i-1}\right)$$

Εδώ όμως η τιμή y'_{i+1} είναι άγνωστη, άρα η μέθοδος **(ii)** δεν είναι εφαρμόσιμη.

$$y'^{(p)}_{i+1} = y'_{i-3} + \frac{4h}{3}\left[2f(x_i, y_i, y'_i) - f(x_{i-1}, y_{i-1}, y'_{i-1}) + 2f(x_{i-2}, y_{i-2}, y'_{i-2})\right]$$

$$y'^{(c)}_{i+1} = y'_{i-1} + \frac{h}{3}\left[f(x_{i+1}, y_{i+1}, y'_{i+1}) + 4f(x_i, y_i, y'_i) + f(x_{i-1}, y_{i-1}, y'_{i-1})\right]$$