Eötvös Loránd Tudományegyetem
Természettudományi Kar

Alkalmazott Analízis és Számításmatematikai
Tanszék

Közönséges Differenciálegyenletek Kezdetiérték Feladatainak Numerikus Megoldása Matlab Alkalmazásával

Szakdolgozat

Készítette:

Macsotai Ágnes

matematika BSc

elemző szakirányos hallgató

Témavezető:

Dr. Faragó István

egyetemi tanár

Budapest

2010

Tartalomjegyzék

Bevezetés

Szakdolgozatom a közönséges differenciálegyenletek numerikus megoldásával foglalkozik. De mik azok a differenciálegyenletek és miért is olyan fontosak? Differenciálegyenlet alatt egy olyan speciális függyényegyenletet értünk, amelyben a meghatározandó ismeretlen egy megfelelő rendben differenciálható függvény, és az egyenlet kapcsolatot fejez ki az ismeretlen függvény különböző rendű deriváltjai között. A differenciálegyenlet megoldása azt jelenti, hogy találunk egy olyan függvényt (függvényeket), amely kielégíti az egyenletet. A fizika, kémia, biológia vagy akár a közgazdaságtan számos alaptörvénye felírható differenciálegyenletekkel. Nézzünk néhány példát!

Fizikai példa: Newton második törvénye azt mondja, hogy az elmozdulás idő szerinti második deriváltja egyenesen arányos a testre ható erővel. Ha az erő minden időpillanatban csak a test helyzetétől függ, akkor a differenciálegyenlet: $x''(t) = m \cdot x(t)$ alakú, ahol a $x(t)$ az ismeretlen függvény. Ezt a másodrendű differenciálegyenletet át tudjuk írni elsőrendűvé a következőképpen: legyen $x_1(t) = x(t)$ és $x_2(t) = x'(t)$, ekkor a differenciálegyenletünk a így néz ki:

$$x_1' = x_2,$$

$$x_2' = mx_1.$$

Biológia példa: A nemlineáris differenciálegyenletek fontos szerepet játszanak a populáció dinamikai modellezésben. Vegyünk egy ragadozó-préda modellt. Legyen $x(t)$ egy nyúlpopuláció mérete a t időpontban, míg $y(t)$ egy rókapopuláció mérete. Ismerjük a populációk kezdeti méretét, $x(0)$-at és $y(0)$-at. Ekkor felírhatjuk a következő diffreneciálegyenletrendszert:

$$x'(t) = ax(t) - bx(t)y(t)$$

$$y'(t) = cx(t)y(t) - dy(t),$$

ahol a a nyulak növekedési rátája, d a rókapopuláció halálozási aránya. Mikor egy róka és egy nyúl találkozik, akkor a nyulak számának csökkennie kell és a rókák számának nőnie, ezt fejezi ki b, illetve c. A továbbiakban egy konkrét példában megadjuk a paraméterek egy lehetséges megválasztását és megvizsgáljuk a modellünket.

1. fejezet

Matematikai bevezetés

1.1. Differenciálegyenletek alapvető tulajdonságai

A differenciálegyenleteknek két típusa van, a közönséges differenciálegyenletek (ebben az esetben egyváltozós az ismeretlen függvény) és a parciális differenciálegyenletek (itt pedig többváltozós az ismeretlen függvény).
Differenciálegyenletek néhány fontosabb jellemzői:

- rend: egy differenciálegyenlet rendjén a differenciálegyenletben szereplő legmagasabb rendű derivált rendjét értjük,
- explicit: ha a függvénykapcsolatból explicit kifejezhető a legmagasabb rendű derivált,
- implicit: ha nem explicit, tehát nem tudjuk kifejezni a legmagasabb rendű deriváltat.

Az elsőrendű explicit közönséges differenciálegyenlet alakja tehát a következő:

$$u' = f(t, u),$$

$$u(t_0) = u_0,$$

ahol $u\colon R \to R$ az ismeretlen differenciálható függvény, és $f\colon T \to R$, $T \subset R^2$ adott folytonos függvény, $(t_0, y_0) \in T$ és az $u'(t) = \frac{du(t)}{dt}$ jelölést használjuk.

Tétel: (egzisztenciatétel) Tegyük fel, hogy teljesülnek az alábbi feltételek:

- $f\colon T \to R$, $T \subset R^2$ egy tartomány,

- $f \in C(T)$, azaz f folytonos T-n,
- f a második változójában eleget tesz a Lipschitz-feltételnek, azaz létezik olyan $L \geq 0$ állandó, hogy minden $(t, u_1), (t, u_2) \in T$-re igaz, hogy

$$|f(t, u_1) - f(t, u_2)| \leq L \, |u_1 - u_2| \, .$$

Ekkor a kezdeti feltételbeli t_0 pontnak létezik olyan $K(t_0)$ környezete, hogy a kezdetiérték feladatnak egyértelműen létezik megoldása $K(t_0)$-n.

1.2. Numerikus módszer

Körülöttünk lévő világunk jobb megismerése céljából a különböző jelenségeket próbáljuk meg modellezni. Mivel világunk bonyolultan működik, így modelljeinket egyszerűsítenünk kell, hogy azokat tesztelni tudjuk (például szabadesésnél eltekinthetünk a közegellenállástól, hogy egyszerűbben ki tudjuk számolni a testre ható erőt). Ilyen folytonos modellek diszkretizációjával foglalkozik a numerikus analízis és az ezeket megoldó numerikus módszerekkel. Természetesen a megoldások közelítőek, de megköveteljük, hogy bizonyos elfogadható és kontrolálható hibahatáron belül maradjanak.
A modellekkel szembeni elvárások:

- létezik megoldás (egzisztencia),
- a megoldás egyértelmű (unicitás),
- a megoldás folyamatosan függ a feladatot leíró adatoktól (stabilitás).

Ezt összefoglalóan korrekt kitűzésű feladatnak nevezzük.

2. fejezet

Egylépéses módszerek

Tekintsük a következő kezdetiérték feladatot:

$$u' = f(t, u), \tag{2.1}$$

$$u(t_0) = u_0. \tag{2.2}$$

Tegyük fel, hogy létezik $u : I \to R$ megoldás, azaz $u \in C^1(I)$ olyan függvény, melyre $u'(t) = f(t, u(t))$, $u(t_0) = u_0$. Ehhez először definiáljuk ω_h rácshálót ily módon:
Legyen $h > 0$ esetén $\omega_h := \{t_n = n \cdot h, n = 0, 1, 2...\}$. Jelölje y_h az ω_h-n értelmezett úgynevezett rácsfüggvényt.
Vezessünk be néhány egyszerűsítő jelölést. Legyen $y_n = y_h(t_n)$ a közelítő érték, $u_n = u(t_n)$ pedig a pontos megoldás a rácsháló t_n pontjában. Feladatunk: úgy meghatározni y_h rácsfüggvényt, mely a rácsháló pontjaiban közel legyen az u függvény értékeihez.

Hibafüggvény: $u(t)$ a kezdetiérték feladat pontos megoldása és y_n a numerikus megoldása, $z_n := y_n - u_n$, $z : \omega_h \to R$ függvény a hibafüggvény.

Definíció: Azt mondjuk, hogy egy numerikus módszer által előállított $y_h(t)$ rácsfüggvénysorozat finomodó h lépésközök esetén konvergál az $u(t)$ kezdetiérték feladat megoldásához, ha valamely $t^* \in I$ pontban

- t^* mindegyik rácshálónak pontja, azaz $t^* \in \omega_h$
- $$\lim_{h \to 0} |y_h(t^*) - u(t^*)| = 0. \tag{2.3}$$

Megjegyzés: A (2.3) esetén azt mondjuk, hogy a numerikus módszer konvergens a t^* pontban. Általánosan a numerikus módszert konvergensnek nevezzük, ha az konvergens minden $t^* \in I$ pontban.

Definíció: Ha egy numerikus módszer konvergens, akkor az (2.3)-beli nullához tartásnak rendjét

$$|y_h(t^*) - y(t^*)| = \mathscr{O}(h^p) \Leftrightarrow |z_h(t^*) = \mathscr{O}(h^p)|$$

a konvergencia rendjének nevezzük.

2.1. Az explicit Euler-módszer

Ismerjük az $u' = f(t, u)$ kezdetiérték feladatot és az u_0 értékét, így ismerjük $u(t)$ deriváltját a t_0 pontban: $u'(t_0) = f(t_0, u(t_0))$, és legyen a kezdeti pontban $y'(t_0) = u'(t_0)$. Egy megfelelően kicsi $h > 0$-t választva a derivált irányában teszünk egy h vetületű lépést.
Az első lépés után: $t_1 = t_0 + h$, $y_1 = y_{t_0} + h \cdot f(t_0, y_{t_0})$
Ekkor kapunk egy új (t_1, y_1) pontot, ebből a pontból újabb h vetületű lépést teszünk a $f(t_1, y_1)$ irányban.
Ezt folytatva kapjuk az explicit Euler-módszert:

$$y_{n+1} = y_n + h \cdot f(t_n, y_n), \tag{2.4}$$

$$y_0 = u_0. \tag{2.5}$$

Most pedig ellenőrizzük az explicit Euler-módszer konvergens-e és nézzük meg, hogy milyen rendben konvergens!
Az egyszerűség kedvéért legyen $u_n = u(t_n)$. A hibafüggvénybe helyettesítsük be az Euler eljárást, ekkor a következő hibaegyenletet kapjuk:

$$\frac{z_{n+1} - z_n}{h} = \left[-\frac{u_{n+1} - u_n}{h} + f(t_n, u_n) \right] + \left[f(t_n, u_n + z_n) - f(t_n, u_n) \right]. \tag{2.6}$$

A $\psi_n^{(1)} := -\frac{u_{n+1} - u_n}{h} + f(t_n, u_n)$ kifejezést a lokális approximációs hibának vagy más néven a reziduális hibának nevezzük. A hibaegyenletből ez a tag fejezi ki, hogy a numerikus módszer milyen pontosan approximálja a folytonos kezdetiérték feladatot, mivel a másik tag csak f-től függ.

Definíció: Az explicit Euler-módszert konzisztensnek nevezzük, ha

$$\lim_{h\to 0} \psi_n^{(1)} = 0. \tag{2.7}$$

A (2.7)-beli konvergencia rendjét a numerikus módszer konzisztencia rendjének nevezzük. Ha $\psi_n^{(1)} = \mathscr{O}(h^p)$, akkor a módszert p-ed rendűnek nevezzük.

2.1.1. Az explicit Euler-módszer konzisztenciája

Az explicit Euler-módszer lokális approximációs hiba egyenletéből a következőt kapjuk:

$$\psi_n^{(1)} := -\frac{u(t_{n+1}) - u(t_n)}{h} + f(t_n, u(t_n)).$$

Írjuk át $u(t_{n+1})$-et $u(t_n+h)$-ra, feltesszük, hogy $u \in C^3(I)$ és Taylor-sorba fejtsük $u(t_n+h)$ kifejezést a t_n pont körül

$$-\frac{u(t_n+h) - u(t_n)}{h} = -\frac{u(t_n) + h\cdot u'(t_n) + \frac{h^2}{2}u''(t_n) + \mathscr{O}(h^3) - u(t_n)}{h},$$

$f(t, u(t))$ helyett írjunk $u'(t)$, majd egyszerűsítünk. Így a következőt kapjuk:

$$\psi_n^{(1)} = -\frac{1}{2}hu''(t_n) + \mathscr{O}(h^2).$$

Tehát az explicit Euler-módszer konzisztens és rendje egy.

2.1.2. Az explicit Euler-módszer konvergenciája

Definíció: Egy numerikus módszer stabilnak nevezünk, ha $f \in Lip(y)$ és létezik olyan $K > 0$ állandó, amely mellett:

$$z_{n+1} \le K\left(|z_0| + h\sum_{i=0}^{n}\left|\psi_i^{(1)}\right|\right).$$

Tétel: Ha az explicit Euler-módszer konzisztens és stabil, akkor konvergens is, és a konvergencia rendje egybeesik a konzisztencia rendjével.

Bizonyítás: Feljebb már szerepelt az explicit Euler-módszer hibaegyenlete, ami a következő:

$$\frac{z_{n+1} - z_n}{h} = \left[-\frac{u_{n+1} - u_n}{h} + f(t_n, u_n)\right] + \left[f(t_n, u_n + z_n) - f(t_n, u_n)\right],$$

Alkalmazva a

$$\psi_n^{(1)} := \left[-\frac{u_{n+1} - u_n}{h} + f(t_n, u_n) \right],$$
$$\psi_n^{(2)} := [f(t_n, u_n + z_n) - f(t_n, u_n)]$$

jelöléseket, rendezzük át a hibaegyenletet a következőképpen:

$$z_{n+1} = z_n + h\psi_n^{(1)} + h\psi_n^{(2)}.$$

Felhasználva a f Lipschitz-tulajdonságát a következő becslés írható fel:

$$|z_{n+1}| \leq |z_n| + hL\,|z_n| + h\left|\psi_n^{(1)}\right| = (1 + hL)\,|z_n| + h\left|\psi_n^{(1)}\right|,$$

ezt a becslést továbbfejtve kapjuk a következőt:

$$|z_n| \leq (1 + hL)\,|z_{n-1}| + h\left|\psi_{n-1}^{(1)}\right| \leq$$
$$\leq (1 + hL)\left[(1 + hL)\,|z_{n-2}| + h\left|\psi_{n-2}^{(1)}\right|\right] + h\left|\psi_{n-1}^{(1)}\right| =$$
$$= (1 + hL)^2\,|z_{n-2}| + (1 + hL)h\left|\psi_{n-2}^{(1)}\right| + h\left|\psi_{n-1}^{(1)}\right| =$$
$$= (1 + hL)^2\,|z_{n-2}| + h\left[(1 + hL)\left|\psi_{n-2}^{(1)}\right| + \left|\psi_{n-1}^{(1)}\right|\right] \leq$$
$$\leq \ldots \leq (1 + hL)^n\,|z_0| + h\sum_{i=0}^{n-1}(1 + hL)^i\left|\psi_{n-i-1}^{(1)}\right|.$$

Most becsüljük felül $1 + hL$-et e^{hL}-vel:

$$|z_n| \leq e^{hLn}\,|z_0| + e^{hLn}h\sum_{i=0}^{n-1}\left|\psi_{n-i-1}^{(1)}\right| \leq e^{hLn}\left[|z_0| + h\sum_{i=0}^{n-1}\left|\psi_{n-i-1}^{(1)}\right|\right],$$

mivel $hn = t^*$ így:

$$= e^{Lt^*}\left[|z_0| + h\sum_{i=0}^{n-1}\left|\psi_{n-i-1}^{(1)}\right|\right] \leq e^{Lt^*}\left[|z_0| + h\frac{M_2}{2}hn\right] = e^{Lt^*}\left[|z_0| + \frac{M_2}{2}t^*h\right],$$

ahol M_2 a maximuma u''-nek a $[0, t^*]$ intervallumon. Azt tudjuk, hogy $z_0 = 0$, mivel a kezdeti pontban ismerjük a pontos megoldást. Így

$$|z_n| \leq e^{Lt^*}\frac{M_2}{2}t^*h = konstans \cdot h.$$

Tehát az explicit Euler-módszer stabil és konzisztens, így konvergens is és a konvergencia rendje egy.

Megjegyzés: A tétel általánosan is igaz, tehát, ha egy numerikus módszer konzisztens és

stabil, akkor konvergens is, és a konvergencia rendje egybeesik a konzisztencia rendjével. A további módszereknél szintén érvényes a stabilitás (ezt a továbbiakban nem számoljuk ki), ezért elég lesz csak a konzisztencia rendjét meghatározni, amiből a konvergencia rendje közvetlenül adódik.

2.1.3. Az explicit Euler-módszer megvalósítása Matlabbal

A Matlabban léteznek már megírt, ún. beágyazott algoritmusok a közönséges differenciálegyenletek megoldására. Mivel az explicit Euler-módszer nincs benne a már megírtak között, így nekünk kell megírni ezt az algoritmust egy m-fájlban. Először is indítsuk el a Matlabot, majd az Editorba írjuk a következőket, majd mentsük el.

```
function [t, y] = eeuler(diffegy, t0, y0, h, N)
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = t0;
y(1) = y0;
for i=1:N
   t(i+1) = t(i) + h;
   y(i+1) = y(i) + h * diffegy(t(i),y(i));
end
```

Az első sorban azt adtuk meg, hogy hogyan fogjuk meghívni a módszerünket. Mi eeuler-nek neveztük el, öt bemenő és kettő kimenő paramétere van. A két kimenő paraméter az idő vektor és a közelített értékek vektora. Az első bemenő paraméter egy alfüggvény, aminek jelen esetben diffegy a neve, ebben a függvényben írjuk meg azt differenciál függvény, amit szeretnénk oldani. A második paraméter t0, ez a kezdeti időpontot jelöli, a harmadik y0, ami a kezdetiérték, a következő h, a lépéstávolság, majd végül N jelöli, hogy hány lépést teszünk meg. A második és harmadik sorban vesszük fel t és y vektorokat, amikben az értékeket először mind 0-ra állítjuk. A következő két sorban beállítjuk a kezdőértékeket. Utána egy ciklus következik, melyben először beállítjuk a t_i értékeket, majd kiszámoljuk a meredekséget, végül az y_i értékeket is, úgy, ahogy az az explicit Euler-módszerben van.

Ez még csak az explicit Euler-módszer megvalósítása, még nem tudjuk egy adott

differenciálegyenletre lefuttatni. Ehhez szükségünk van egy alfüggvényre, aminek a neve diffegy lesz. A következő példát fogjuk leprogramozni:

$$u'(t) = -u(t) + t + 1,$$

ahol $t \in [0, 1]$ és $u(0) = 1$. Kétféle lépéstávolságot fogunk vizsgálni, $h_1 = 0.1$-et és $h_2 = 0.01$-et. A példánkat természetesen exact módon is ki lehet számolni, hogy meg tudjuk nézni, mennyire pontos a módszer.

Most pedig írjuk meg a diffegy nevű alfüggvényünket. Nyissunk egy új m-fájlt, amibe írjuk a következőket, majd mentsük el.

```
function dydt = diffegy(t,y)
dydt = -y + t + 1;
```

Most már megírtunk mindkét függvényt, futtassuk le a programunkat először $h_1 = 0.1$-es lépéstávolsággal, majd $h_2 = 0.01$-es lépéstávolsággal is. A Command ablakba írjuk a következőt: `[T1,Ye] = eeuler(@diffegy, 0, 1, 0.1, 10)`. Enter után kiadja T1 és Ye vektorokat. Ha ki is szeretnénk rajzoltatni, akkor a `plot(T1,Ye)` paranccsal egy külön ablakban megjelenik a függvényünk. A $h_2 = 0.01$-es lépéstávolsághoz pedig írjuk ezt: `[T2,Xe] = eeuler(@diffegy, 0, 1, 0.01, 100)`.

Mivel az explicit Euler-módszer hibájára vagyunk kíváncsiak, a pontos megoldást is le kell programoznunk. A példánk pontos megoldása:

$$u(t) = e^{-t} + t$$

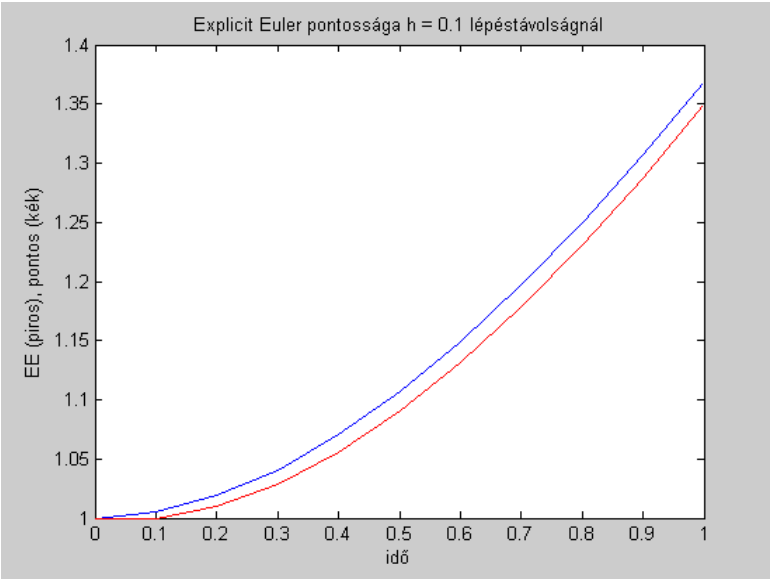
függvény. A Matlabban egy új m-fájlba írjuk a következőket:

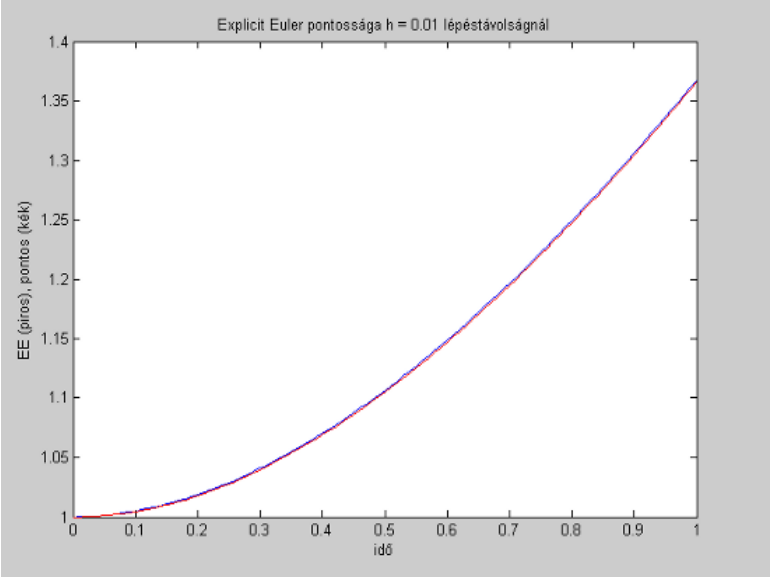
```
function e = exact(t)
e = exp(-t) + t;
```

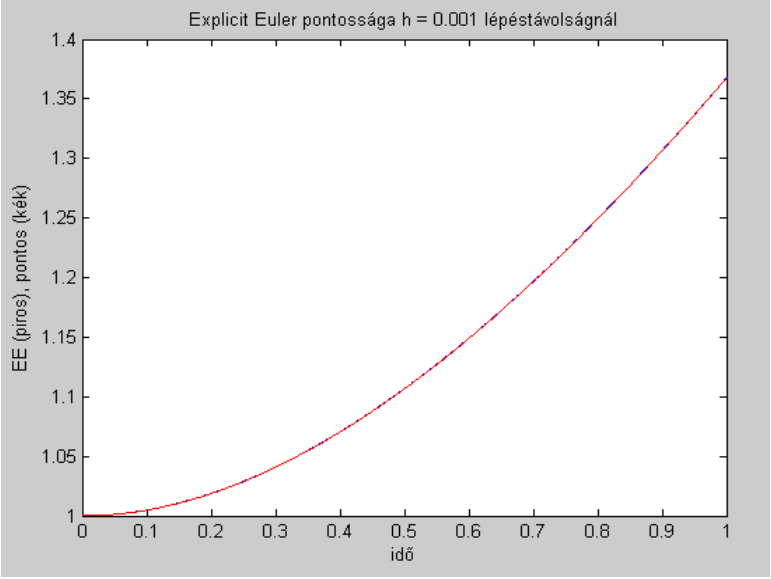
A Command ablakban hívjuk meg. Legyen Yp a pontos megoldás T1 intervallumon, míg Xp a pontos megoldás T2 intervallumon. A `plot(T1, [Ye, Yp])` és `plot(T2, [Xe, Xp])` parancsokkal egy ábrán is megtekinthetjük a pontos és az approximált függvényeket. Az alábbi táblázatban láthatjuk, a módszer hibáját $h_1 = 0.1$ és $h_2 = 0.01$ lépéstávolságoknál.

t_i	$u(t_i)$	$y(t_i)$	z_i
0	1.0000	1.0000	0
0.1000	1.0048	1.0000	4.8374e-003
0.2000	1.0187	1.0100	8.7308e-003
0.3000	1.0408	1.0290	1.1818e-002
0.4000	1.0703	1.0561	1.4220e-002
0.5000	1.1065	1.0905	1.6041e-002
0.6000	1.1488	1.1314	1.7371e-002
0.7000	1.1966	1.1783	1.8288e-002
0.8000	1.2493	1.2305	1.8862e-002
0.9000	1.3066	1.2874	1.9149e-002
1.0000	1.3679	1.3487	1.9201e-002

t_i	$u(t_i)$	$y(t_i)$	z_i
0	1.0000	1.0000	0
0.0100	1.0000	1.0000	4.9834e-005
0.0200	1.0002	1.0001	9.8673e-005
⋮	⋮	⋮	⋮
0.1000	1.0048	1.0044	4.5534e-004
⋮	⋮	⋮	⋮
0.5000	1.1065	1.1050	1.5246e-003
⋮	⋮	⋮	⋮
0.8000	1.2493	1.2475	1.8058e-003
⋮	⋮	⋮	⋮
0.9800	1.3553	1.3535	1.8468e-003
0.9900	1.3616	1.3597	1.8471e-003
1.0000	1.3679	1.3660	1.8471e-003


Explicit Euler pontossága h = 0.1 lépéstávolságnál
EE (piros), pontos (kék)
idő


Explicit Euler pontossága h = 0.01 lépéstávolságnál
EE (piros), pontos (kék)
idő


Explicit Euler pontossága h = 0.001 lépéstávolságnál
EE (piros), pontos (kék)
idő

Amint azt láthatjuk, a minél kisebb lépéstávolságot használunk, annál pontosabb lesz a közelítésünk a $t^* = 1$ pontban.

h	0.1	0.01	0.001
z_{i^*}	1.9201e-002	1.8471e-003	1.8402e-004

2.1.4. Lotka-Volterra Modell

A bevezetőben már említett ragadozó-préda vagy más néven Lotka-Volterra modellt fogjuk ebben a részben Matlabban megvalósítani. Legyen $x(t)$ a préda populáció mérete a t időpontban, míg $y(t)$ a ragadozó populáció mérete. Ha nincs ragadozó, akkor feltesszük, hogy a préda populáció exponenciális vagy $x' = ax$ dinamikájú, ahol a a préda populáció növekedési rátája. Feltesszük még, hogy préda hiányában a ragadozók kihalnak $y' = -dy$ egyenlet szerint, ahol d a ragadozó populáció halálozási aránya. Mikor préda ragadozóval találkozik, a préda populációnak csökkennie kell $x'(t) = ax(t) - bx(t)y(t)$ és a ragadozó populációnak növekednie $y'(t) = cx(t)y(t) - dy(t)$. Ismerjük a populációk kezdeti méretét, $x(0)$-at és $y(0)$-at, mely a két populáció kezdeti arányát mutatja meg. Ekkor felírhatjuk a következő differenciálegyenlet-rendszert:

$$x'(t) = ax(t) - bx(t)y(t)$$

$$y'(t) = cx(t)y(t) - dy(t).$$

Nézzük a következő példát:

$$x' = x - 2x^2 - xy$$

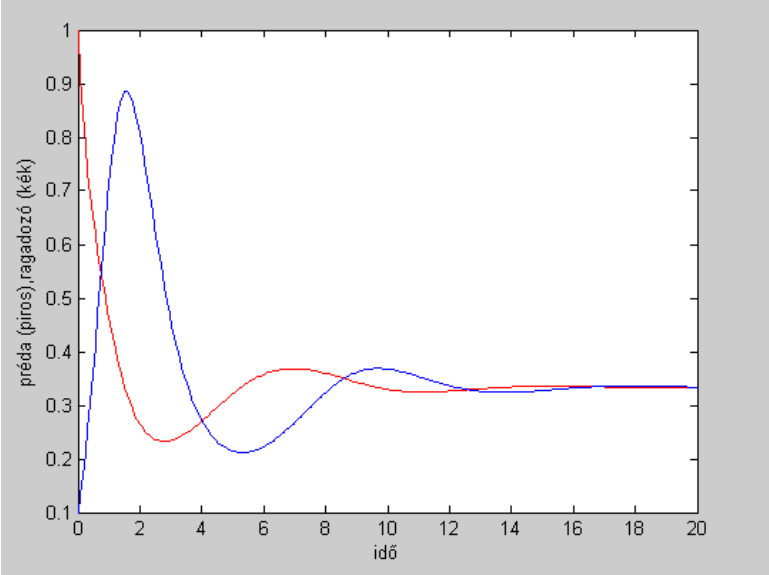
$$y' = -2y + 6xy,$$

legyen $x(0) = 1$, $y(0) = 0.1$ és a $t = [0, 20]$. Mathlabban nyissunk egy új m-fájlt és írjuk bele a következőket:

```
function euler (x, y, T, N)
xhistory=x; yhistory=y; h=T/N;
for n=1:N
   u=f(x,y); v=g(x,y);
   x=x+h*u; y=y+h*v;
   xhistory=[xhistory,x]; yhistory=[yhistory,y];
end
```

```
t=0:h:T;
plot(t,xhistory,'red', t, yhistory,'blue')
xlabel('idő'), ylabel('préda (piros), ragadozó (kék)')
function U=f(x,y)
U=x-2*x.*x-x.*y;
function V=g(x,y)
V=-2*y+6*x.*y;
```

Mentsük el, futtassuk így: `euler(1, 0.1, 20, 250)` után ezt a grafikont kapjuk:



Ahogy az a valóságban is van, a populáció vagy kihal, vagy egyensúlyba kerül. A grafikonon is jól látszik, hogy a kezdeti időpontokban a ragadozók száma csökken, mivel nincsenek préda állatok, ezzel egy időben a prédaállatok száma nő és, ahogy telik az idő úgy kerülnek egyensúlyba a populáció méretei.

2.2. A javított Euler-módszer

Próbáljuk meg javítani az Euler-módszert úgy, hogy ne elsőrendű, hanem másodrendű legyen. Először egy fél lépést téve kiszámítjuk $f(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})$ értékét, majd ezzel az új meredekséggel teszünk egy egész lépést az eredeti (t_n, y_n) pontból.

A módszer algoritmikus leírása a következő:

$$y_{n+1} = y_n + h \cdot f(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}}). \tag{2.8}$$

2.2.1. A javított Euler-módszer rendje

A lokális approximációs hiba kiszámolásához fel kell tennünk, hogy f mindkét változója szerint kétszer folytonosan differenciálható, azaz $f \in C^2(T)$.

$$\psi_n^{(1)} = -\frac{u_{n+1} - u_n}{h} + f(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})$$

Az u_{n+1} és $u'(t_{n+\frac{1}{2}}) = f(t_{n+\frac{1}{2}}, y(t_{n+\frac{1}{2}}))$ függvényeket sorba fejtve a t_n pont körül a kapjuk:

$$\psi_n^{(1)} = -\frac{u(t_n) + hu'(t_n) + \frac{h^2}{2}u''(t_n) + \mathcal{O}(h^3) - u(t_n)}{h}$$

$$+u'(t_n) + \frac{h}{2}u''(t_n) + \frac{h^2}{8}u'''(t_n) + \mathcal{O}(h^3).$$

A zárójel felbontása és a h-val való osztás után:

$$= -u'(t_n) - \frac{h}{2}u''(t_n) + o(h^2) + u'(t_n) + \frac{h}{2}u''(t_n) + \frac{h^2}{8}u'''(t_n) + \mathcal{O}(h^3)$$

$$= \frac{h^2}{8}u'''(t_n) + \mathcal{O}(h^2).$$

Tehát a javított Euler-módszer másodrendben konzisztens, ahogy vártuk.

2.2.2. A javított Euler-módszer megvalósítása Matlabbal

Az explicit Euler-módszerhez hasonlóan a javított Euler-módszer sincs a beágyazott algoritmusok között, ezt is nekünk kell megírnunk. Nyissunk egy új m-fáljt, amibe írjuk a következőket:

```
function [t, y] = javeuler(diffegy, t0, y0, h, N)
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = t0;
y(1) = y0;
hfel = 0.5 * h;
for i=1:N
   t(i+1) = t(i) + h;
    y(i+1) = y(i) + h*diffegy(t(i)+hfel, y(i)+ hfel*diffegy(t(i), y(i)));
end
```

A program nem sokban tér el az explicit Euler-módszernél használttól. A különbség az az érték, mikor fél lépést teszünk, ehhez csak a meredekséget kell megváltoztatnunk. Először is a cikluson kívül még definiálnunk kell $hfel = h + 0,5$ -t, és a ciklusban a meredekség definícióját is megfelelően módosítjuk.

Most is ugyanazt a példát vizsgáljuk $h_1 = 0.1$ és $h_2 = 0.01$ lépéstávolságokkal. A Command ablakban a következőképpen hívjuk meg `[T1, Yj] = javeuler(@diffegy, 0, 1, 0.1, 10)` és `[T2, Xj] = javeuler(@diffegy, 0, 1, 0.01, 100)`. A `plot(T1, [Yj, Yp])` és a `plot(T2, [Xj, Xp])` parancsokkal megint ki tudjuk rajzoltatni a közelítő és a pontos megoldást. Az alábbi táblázatban láthatjuk a pontos értékeket, a javított Euler-módszer közelítéseit és hibáját $h_1 = 0.1$ és $h_2 = 0.01$ lépéstávolságoknál.

t_i	$u(t_i)$	$y(t_i)$	z_i
0	1.0000	1.0000	0
0.1000	1.0048	1.0050	1.6258e-004
0.2000	1.0187	1.0190	2.9425e-004
0.3000	1.0408	1.0412	3.9940e-004
0.4000	1.0703	1.0708	4.8190e-004
0.5000	1.1065	1.1071	5.4511e-004
0.6000	1.1488	1.1494	5.9193e-004
0.7000	1.1966	1.1972	6.2492e-004
0.8000	1.2493	1.2500	6.4629e-004
0.9000	1.3066	1.3072	6.5795e-004
1.0000	1.3679	1.3685	6.6154e-004

t_i	$u(t_i)$	$y(t_i)$	z_i
0	1.0000	1.0000	0
0.0100	1.0000	1.0001	1.6625e-007
0.0200	1.0002	1.0002	3.2919e-007
⋮	⋮	⋮	⋮
0.1000	1.0048	1.0048	1.5194e-006
⋮	⋮	⋮	⋮
0.5000	1.1065	1.1065	5.0925e-006
⋮	⋮	⋮	⋮
0.8000	1.2493	1.2493	6.0362e-006
⋮	⋮	⋮	⋮
0.9000	1.3066	1.3066	6.1445e-006
⋮	⋮	⋮	⋮
0.9900	1.3616	1.3616	6.1772e-006
1.0000	1.3679	1.3679	6.1775e-006

	EE	EE	jav E	jav E
h	0.1	0.01	0.1	0.001
z_{n^*}	1.9201e-002	1.8471e-003	6.6154e-004	6.1775e-006

A példán is jól láthatjuk, hogy a javított Euler-módszer gyorsabban konvergál, mint az explicit Euler-módszer.

2.3. Az implicit Euler-módszer

Próbáljuk meg még jobban javítani az Euler-módszer. Most ne a (t_n, y_n) meredekséggel lépjünk, hanem a (t_{n+1}, y_{n+1}) meredekséggel. Ekkor a módszerünk implicit lesz, és azt reméljük, hogy magasabb rendű, mint az explicit Euler-módszer.
Módszer leírása képlettel:

$$y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1}), \tag{2.9}$$

$$y_0 = u_0. \tag{2.10}$$

2.3.1. Az implicit Euler-módszer konzisztenciája

Az explicit Euler-módszerhez hasonlóan számolhatjuk ki az implicit módszer rendjét.

$$\psi_n^{(1)} := -\frac{u(t_{n+1}) - u(t_n)}{h} + f(t_{n+1}, u(t_{n+1})),$$

mivel $f(t_{n+1}, u(t_{n+1})) = u'(t_{n+1})$-gyel, így

$$= -\frac{u(t_{n+1}) - u(t_{n+1} - h)}{h} + u'(t_{n+1}).$$

Ha $u(t_{n+1} - h)$ kifejezést a t_{n+1} körül Taylor-sorba fejtjük, a következőt kapjuk:

$$\psi_n^{(1)} = -\frac{u(t_{n+1}) - (u(t_{n+1}) - h \cdot u'(t_{n+1}) + \mathcal{O}(h^2))}{h} + u'(t_{n+1}),$$

a zárójelet felbontva és egyszerűsítve:

$$= \frac{u(t_{n+1}) + u(t_{n+1}) - h \cdot u'(t_{n+1}) + \mathcal{O}(h^2)}{h} = \mathcal{O}(h).$$

Tehát az implicit Euler-módszer, a várttal ellentétben az explicit módszerhez hasonlóan első rendben konzisztens.

2.3.2. Az implicit Euler-módszer megvalósítása Matlabbal

Az implicit feladatok leprogramozása általában bonyolult, így én két egyszerűbb megoldást fogok bemutatni a fentebb említett példánkra. Elsőnek nézzük azt a megoldást, ami pontosabb, de csak erre a példára alkalmazható. Ehhez először is kicsit át kell írnunk a feladatunkat. Tehát az $u'(t) = -u(t) + t + 1$ egyenletet írjuk be az implicit Euler-módszerbe:

$$y_{n+1} = y_n + h \cdot (-y_{n+1} + t_{n+1} + 1).$$

Most bontsuk fel a zárójelet, majd rendezzük át:

$$y_{n+1} = y_n - h \cdot y_{n+1} + h \cdot t_{n+1} + h,$$

$$y_{n+1} = \frac{y_n + h \cdot t_{n+1} + h}{1 + h}.$$

Ezt programozzuk le a már megszokott módon. Nyissunk egy új m-fájlt, amibe írjuk az alábbiakat:

```
function [t, y] = ieuler(t0, y0, h, N)
```

```
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = t0;
y(1) = y0;
for i=1:N
   t(i+1) = t(i) + h;
    y(i+1) = (y(i) + h * t(i+1) + h) / (1+h);
end
```

Mentsük el, majd a Command ablakban hívjuk meg így: `[T1, Yi] = ieuler(0, 1, 0.1, 10)`. Majd $h_2 = 0.01$ lépéstávolsággal is futtassuk le. A módszer hibáját a táblázatban láthatjuk $h_1 = 0.1$ és $h_2 = 0.01$ lépéstávolságok mellett.

t_i	$u(t_i)$	$y(t_i)$	z_i
0	1.0000	1.0000	0
0.1000	1.0048	1.0091	4.2535e-003
0.2000	1.0187	1.0264	7.7155e-003
0.3000	1.0408	1.0513	1.0497e-002
0.4000	1.0703	1.0830	1.2693e-002
0.5000	1.1065	1.1209	1.4391e-002
0.6000	1.1488	1.1645	1.5662e-002
0.7000	1.1966	1.2132	1.6573e-002
0.8000	1.2493	1.2665	1.7178e-002
0.9000	1.3066	1.3241	1.7528e-002
1.0000	1.3679	1.3855	1.7664e-002

t_i	$u(t_i)$	$y(t_i)$	z_i
0	1.0000	1.0000	0
0.0100	1.0000	1.0001	4.9176e-005
0.0200	1.0002	1.0003	9.7376e-005
⋮	⋮	⋮	⋮
0.1000	1.0048	1.0053	4.4954e-004
⋮	⋮	⋮	⋮
0.5000	1.1065	1.1080	1.5082e-003
⋮	⋮	⋮	⋮
0.8000	1.2493	1.2511	1.7890e-003
⋮	⋮	⋮	⋮
0.9000	1.3066	1.3084	1.8215e-003
⋮	⋮	⋮	⋮
0.9900	1.3616	1.3634	1.8316e-003
1.0000	1.3679	1.3697	1.8318e-003

Nézzünk meg egy másik módszert is, ami nem ennyire pontos, de nem kell minden megoldani kívánt differenciálegyenlethez írnunk egy-egy új módszert.

```
function [t, y] = ie(diffegy, t0, y0, h, N)
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = t0;
y(1) = y0;
for i=1:N
   t(i+1) = t(i) + h;
   yexp = y(i) + h * diffegy(t(i),y(i));
   y(i+1) = y(i) + h * diffegy(t(i+1), yexp);
end
```

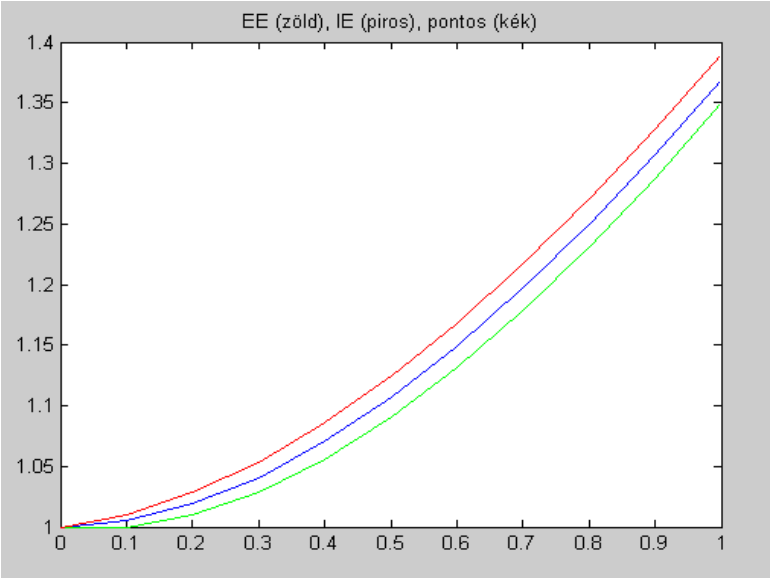
Itt explicit Euler-módszerrel közelítjük y_{n+1} értékét, amit az implicit Euler-módszerben használunk fel, bár így nem lesz annyira pontos a módszer, mint az előbb. Természetesen nem muszáj az explicit Euler-módszert használni, bármelyik másik explicit módszer is jó, hogy approximáljuk y_{n+1}-et. Tulajdonképpen a (2.9) implicit Euler-módszer nemlineáris

egyenletét oldjuk meg az egyszerű iteráció módszerével, amikor is egy iterációs lépést teszünk csak meg és kezdőértéknek az explicit Euler-módszer eredményét használjuk. Mentsük el az m-fájlt, majd hívjuk meg a következőképpen: `[T1, Yie] = ie(@diffegy, 0, 1, 0.1, 10)`, majd $h_2 = 0.01$ lépéstávolsággal is: `[T2, Xie] = ie(@diffegy, 0, 1, 0.01, 100)`. A módszer hibáját az alábbi táblázatokban láthatjuk:

t_i	$u(t_i)$	$y(t_i)$	z_i
0	1.0000	1.0000	0
0.1000	1.0048	1.0100	5.1626e-003
0.2000	1.0187	1.0281	9.3692e-003
0.3000	1.0408	1.0536	1.2753e-002
0.4000	1.0703	1.0857	1.5430e-002
0.5000	1.1065	1.1240	1.7501e-002
0.6000	1.1488	1.1679	1.9058e-002
0.7000	1.1966	1.2168	2.0176e-002
0.8000	1.2493	1.2703	2.0924e-002
0.9000	1.3066	1.3279	2.1360e-002
1.0000	1.3679	1.3894	2.1537e-002

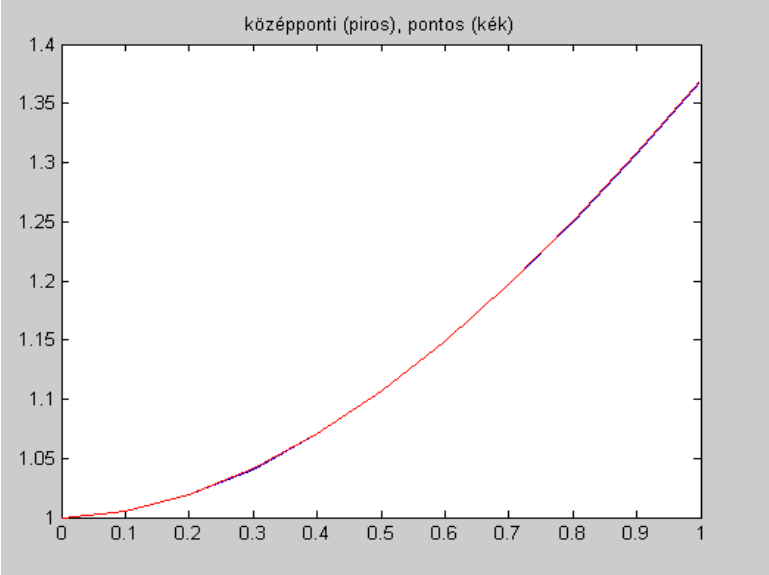
t_i	$u(t_i)$	$y(t_i)$	z_i
0	1.0000	1.0000	0
0.0100	1.0000	1.0001	5.0166e-005
0.0200	1.0002	1.0003	9.9337e-005
⋮	⋮	⋮	⋮
0.1000	1.0048	1.0053	4.5859e-004
⋮	⋮	⋮	⋮
0.5000	1.1065	1.1081	1.5386e-003
⋮	⋮	⋮	⋮
0.8000	1.2493	1.2512	1.8251e-003
⋮	⋮	⋮	⋮
0.9000	1.3066	1.3084	1.8583e-003
⋮	⋮	⋮	⋮
0.9800	1.3553	1.3572	1.8683e-003
0.9900	1.3616	1.3634	1.8686e-003
1.0000	1.3679	1.3697	1.8687e-003

	ieuler	ieuler	ie	ie
h	0.1	0.01	0.1	0.001
z_{i^*}	1.7664e-002	1.8318e-003	2.1537e-002	1.8687e-003



Ha egy grafikonon ábrázoljuk az implicit Euler-módszer, az explicit Euler-módszer és a

pontos megoldásokat, észrevehetjük, hogy az implicit Euler közelítése a pontos megoldás felett, míg az explicit Euler-módszeré a pontos megoldás alatt fut. Ebből az észrevételből jön a középponti szabály, amit tehát úgy származtatunk, hogy az implicit Euler-módszer és az explicit Euler-módszer közelítéseit összeadjuk, majd elosztjuk kettővel. Ekkor egy még jobb közelítést kapunk:



2.4. Runge-Kutta típusú módszerek

Ebben a fejezetben ismerkedhetünk meg a Runge-Kutta módszerekkel, amik szintén az egylépéses módszerekhez tartoznak, azaz y_n-ből y_{n+1}-et számoljuk. Más egylépéses módszerektől eltérően itt az eredmény kiszámolásához előbb több "lépcsőt" is ki kell számolnunk. Így magasabb rendű módszereket is elő tudunk állítani.
Definiáljuk az alábbi m darab k_i számokat a következőképpen:

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + ha_2, y_n + hb_{21}k_1)$$

$$k_3 = f(t_n + ha_3, y_n + h(b_{31}k_1 + b_{32}k_2))$$

$$\vdots$$

$$k_m = f(t_n + ha_m, h \cdot \sum_{i=1}^{m-1} b_{mi}k_i).$$

Ezen k_i számok lineáris kombinációjának segítségével definiáljuk az általános, m-lépcsős Runge-Kutta módszert:

$$y_{n+1} = y_n + h \cdot \sum_{i=1}^{m} \sigma_i k_i.$$

A paramétereket az ún. Butcher-táblázatba rendezzük:

a_1	b_{11}	$\cdots$	b_{1m}
a_2	b_{21}	$\cdots$	b_{2m}
$\vdots$		$\ddots$	
a_m	b_{m1}	$\cdots$	b_{mm}
	σ_1	$\cdots$	σ_m

Az explicit Runge-Kutta módszereknél a mátrix egy szigorúan alsóháromszög mátrix, míg az implicit Runge-Kutta módszereknél nem ilyen. Ha alsóháromszög mátrix, tehát a főátlóban nem csak nulla elemek vannak diagonális implicit Runge-Kutta (DIRK) módszernek nevezzük.

Az explicit Euler-módszer értelmezhető Runge-Kutta módszernek is. A Butcher táblája a következő:

0	0
	1

Az explicit Euler-módszer elsőrendű volt, de nem is tudunk nagyobb rendű, egylépcsős módszert előállítani.

Az implicit Euler-módszer Butcher táblája:

1	1
	1

A javított Euler-módszer Butcher táblája:

0	0	0
0.5	0.5	0
	0	1

A javított Euler-módszer kétlépcsős, másodrendű módszer. Nézzük meg, hogy van-e más kétlépcsős, másodrendű módszer, illetve létezik-e kétlépcsős, harmadrendű módszer!

2.4.1. Kétlépcsős Runge-Kutta módszerek

Általános módszer $m = 2$ esetén:

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + ha_2, y_n + hb_{21}k_1)$$

$$y_{n+1} = y_n + h(\sigma_1 k_1 + \sigma_2 k_2)$$

A megválasztható szabad paramétereink a_2, b_{21}, σ_1 és σ_2. Írjuk fel először a lokális approximációs hiba egyenletét erre az általános esetre.

$$\psi_n^{(1)} = -\frac{u(t_{n+1}) - u(t_n)}{h} + \left[\sigma_1 f(t_n, u(t_n)) + \sigma_2 f(t_n + ha_2, u(t_n) + b_{21}hf(t_n, u(t_n)))\right].$$

Fejtsük Taylor-sorba $u(t_{n+1})$-et t_n pont körül.

$$\frac{u(t_n + h) - u(t_n)}{h} = \frac{u(t_n) + hu'(t_n) + \frac{h^2}{2}u''(t_n) + \mathscr{O}(h^3) - u(t_n)}{h} = u'(t_n) + \frac{h}{2}u''(t_n) + \mathscr{O}(h^2).$$

Most pedig $f(t_n + ha_2, u(t_n) + b_{21}hf(t_n, u(t_n)))$-et kell a $(t_n, u(t_n))$ pont körül.

$$f(t_n + ha_2, u(t_n) + b_{21}hf(t_n, u(t_n))) = f + \partial_1 f \cdot a_2 h + \partial_2 f \cdot b_{21}hf + \mathscr{O}(h^2),$$

ahol f, $\partial_1 f$ és $\partial_2 f$ a $(t_n, u(t_n))$ helyen értendő. Az alábbiakat kaptuk:

$$\psi_n^{(1)} = [-u'(t_n) + \sigma_1 f + \sigma_2 f] + h\left[-\frac{1}{2}u''(t_n) + \sigma_2(a_2\partial_1 f + b_{21}\partial_2 f \cdot f)\right] + \mathscr{O}(h^2)$$

Mivel $u'(t) = f(t, u(t))$ megoldása a differenciálegyenletnek és $u''(t) = \partial_1 f + f\partial_2 f$ a $(t, u(t))$ helyen, így

$$\psi_n^1 = -f + (\sigma_1 + \sigma_2)f + h\left[-\frac{1}{2}(\partial_1 f) - \frac{1}{2}(\partial_2 f)f + a_2\sigma_2(\partial_1 f) + \sigma_2(\partial_2 f)b_{21}f\right] + \mathscr{O}(h^2).$$

Hogy az általános módszerünk másodrendű legyen az alábbi feltételeknek kell teljesülniük:

$$\sigma_1 + \sigma_2 = 1,$$

$$a_2\sigma_2 = \frac{1}{2},$$

$$b_{21}\sigma_2 = \frac{1}{2}.$$

Legyen $\sigma := \sigma_2$. Ekkor $\sigma_1 = 1 - \sigma$ és legyen $a := a_2 = b_{21} = \frac{1}{2\sigma}$, $\sigma \neq 0$ az előbbi feltételek miatt. Tehát az általános kétlépcsős, másodrendű Runge-Kutta módszer táblázata így néz ki:

0	0	0
a	a	0
	$1-\sigma$	σ

ahol $a\sigma = \frac{1}{2}$.

Másik kérdésünk az volt, hogy lehet-e kétlépcsős, harmadrendű módszer előállítani. Tehát a fenti paraméteres módszerben a-t vagy σ-át lehet-e úgy megválasztani, hogy a módszer harmadrendű legyen?

Először is írjuk fel az általános alakot:

$$y_{n+1} = y_n + h\left[(1-\sigma)f(t_n, y_n) + \sigma f(t_n + ah, y_n + ahf(t_n, y_n))\right],$$

ahol a $a\sigma = 0.5$.

Megmutatjuk, hogy nem lehetséges ilyen megválasztás. Ehhez elég azt megmutatnunk, hogy van olyan elsőrendű közönséges differenciálegyenlet, amelyre semmilyen paraméterválasztás mellett sem lehet a kétlépcsős módszer harmadrendű. Tekintsük a következő differenciálegyenletet:

$$u' = u,$$

$$u(0) = u_0.$$

Mivel ebben a feladatban $f(t,u) = u$, így $f(t_n, y_n) = y_n$. Ezen megválasztás esetén az alábbi módon módosul a képletünk.

$$y_{n+1} = y_n + h\left[(1-\sigma)y_n + \sigma(y_n + ahy_n)\right].$$

Elvégezve a beszorzásokat és egyszerűsítéseket, ekkor

$$y_{n+1} = y_n + h(y_n - \sigma y_n + \sigma y_n + ahy_n) = y_n + h(1 + a\sigma h)y_n.$$

Mivel $a\sigma = 0,5$, így:

$$\frac{y_{n+1} - y_n}{h} = (1 + 0,5h)y_n.$$

Most már felírhatjuk az approximációs hibaegyenletét:

$$\psi_n^{(1)} = -\frac{u(t_{n+1}) - u_{t_n}}{h} + (1 + 0,5h)u_n,$$

Taylor-sorba fejtjük $u(t_{n+1})$-et t_n körül:

$$\psi_n^{(1)} = -\left(u'(t_n) + \frac{h}{2}u''(t_n) + \frac{h^2}{6}u'''(t_n) + \mathcal{O}(h^3)\right) + (1 + 0,5h)u(t_n).$$

A differenciálegyenletből következik, hogy $u(t) = u'(t) = u''(t) = u'''(t)$.

$$\psi_n^{(1)} = u(t_n)\left[-1 - \frac{h}{2} - \frac{h^2}{6} + 1 + \frac{1}{2}h\right] + \mathscr{O}(h^3) = -\frac{h^2}{6}u(t_n) + \mathscr{O}(h^3).$$

Mivel $u(t_n) \neq 0$, ezért a módszer legfeljebb másodrendű lehet.

Hogy egy módszer p rendben konzisztens legyen az alábbi feltétel(ek)nek kell teljesülnie:

rend(p)	Feltétel
1	$\sigma e = 1$
2	$\sigma a = \frac{1}{2}$
3	$\sigma a^2 = \frac{1}{3}$, $\sigma Ba = \frac{1}{6}$
4	$\sigma a^3 = \frac{1}{4}$, $\sigma ABa = \frac{1}{8}$, $\sigma Aa^2 = \frac{1}{12}$, $\sigma A^2 a = \frac{1}{24}$

ahol $e = (1, 1, \ldots, 1)$, $a^k = \left[a_1^k, a_2^k, \ldots, a_s^k\right]$, $A = diag[a_1, \ldots, a_s]$ és B a Butcher táblázatból a b_{ij}-k által alkotott mátrix.

Azt már tudjuk, hogy kevesebb lépcső biztosan nem elég magasabb rendű módszerek előállításhoz. Sokszor több lépcsőre van szükségünk, mint a kívánt rend. A következő táblázatban láthatjuk, hogy hogyan kapcsolódik a rend p és a lépcsők m száma egymáshoz.

Lépcsők száma (m)	Rend(p)
$1 \leq m \leq 4$	$p(m) = m$
$5 \leq m \leq 7$	$p(m) \leq m - 1$
$8 \leq m \leq 10$	$p(m) \leq m - 2$
$m > 10$	$p(m) \leq m - 3$

2.4.2. Magasabb rendű Runge-Kutta módszerek a Matlabban

Már említettem, hogy léteznek beágyazott differenciálegyenlet megoldó algoritmusok a Matlabban. Ilyen például az ODE45, ami a Dormand-Prince módszert alkalmazza. Ez egy egylépéses, váltakozó lépésközű módszer. Kiszámol egy negyed és egy ötöd rendű Runge-Kutta módszert, és úgy választ lépésközt, hogy a hiba a negyed rendű módszer hibája legyen. A következő táblázatban láthatjuk a módszer Butcher-tábláját. Az első σ sor a negyed rendű módszer megoldását adja, míg a második az ötöd rendűjét.

0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$\frac{-56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$\frac{-25360}{2187}$	$\frac{64448}{6561}$	$\frac{-212}{729}$			
1	$\frac{9017}{3168}$	$\frac{-355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$\frac{-5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$\frac{-2187}{6784}$	$\frac{11}{84}$	
	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$\frac{-92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$
	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$\frac{-2187}{6784}$	$\frac{11}{84}$	0

Hívjuk meg az ODE45 algoritmust hasonló módon, mint a saját magunk által írt programokat: `[T1, Y45] = ode45(@diffegy, T1, 1)`. Itt is két kimenő paraméter van, az idő és a közelítő érték vektor. A bemenő paramétere rendre: az alfüggvény, amiben a differenciálegyenletünk definíciója van, az idő vektor, hogy melyik időpontokban számoljuk ki a megoldást, a kezdetiérték vektor. Meg lehet adni egy negyedik választható paramétert is (options), aminek segítségével a default integrálási értékeket átállíthatjuk.

Az alábbi táblázatokban láthatjuk, hogy mennyire pontos az ODE45 nemcsak $h_2 = 0.01$, hanem $h_1 = 0.1$ lépéstávolság mellett is.

t_n	$u(t_n)$	$y(t_n)$	z_n
0	1.0000	1.0000	0
0.1000	1.0048	1.0048	2.9737e-010
0.2000	1.0187	1.0187	5.3815e-010
0.3000	1.0408	1.0408	7.3041e-010
0.4000	1.0703	1.0703	8.8120e-010
0.5000	1.1065	1.1065	9.9668e-010
0.6000	1.1488	1.1488	1.0822e-009
0.7000	1.1966	1.1966	1.1424e-009
0.8000	1.2493	1.2493	1.1814e-009
0.9000	1.3066	1.3066	1.2026e-009
1.0000	1.3679	1.3679	1.2090e-009

t_n	$u(t_n)$	$y(t_n)$	z_n
0	1.0000	1.0000	0
0.0100	1.0000	1.0000	2.2204e-016
0.0200	1.0002	1.0002	8.8940e-011
⋮	⋮	⋮	⋮
0.1000	1.0048	1.0048	5.4080e-009
⋮	⋮	⋮	⋮
0.5000	1.1065	1.1065	2.8278e-009
⋮	⋮	⋮	⋮
0.8000	1.2493	1.2493	1.6519e-009
⋮	⋮	⋮	⋮
0.9000	1.3066	1.3066	1.3610e-009
⋮	⋮	⋮	⋮
0.9900	1.3616	1.3616	1.0920e-009
1.0000	1.3679	1.3679	1.0903e-009

A módszer pontossága igen keveset javult $h_2 = 0.01$ lépéstávolságnál. Ez egyrészt azért van, mert az ODE45 algoritmus változó lépésközű, másrészt van egy beépített default érték a pontosságra is (ezt az értéket már $h_1 = 0.1$-nél elérte). A negyedik bemenő paraméterrel állíthatjuk át a módszer pontosságát az ODESET függvény segítségével vagy mi magunk is írhatunk ilyen függvényeket.

Nézzünk egy másik beágyazott módszert is, nevezetesen az ODE23, ami szintén Runge-Kutta módszereken alapszik, mégpedig a Bogacki-Shampine módszeren.
Bogacki-Shampine Butcher-táblája:

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{3}{4}$	0	$\frac{3}{4}$		
1	$\frac{2}{9}$	$\frac{1}{3}$	$\frac{4}{9}$	
	$\frac{2}{9}$	$\frac{1}{3}$	$\frac{4}{9}$	0
	$\frac{7}{24}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{8}$

Most futtassuk le a példánkra az ODE23 algoritmust is. Először $h_1 = 0.1$ lépéstávolsággal: `[T1, Y23] = ode23(@diffegy, T1, 1)`, majd $h_2 = 0.01$ lépéstávolsággal: `[T2,`

`X23] = ode23(@diffegy, T2, 1)`. A következő táblázatok mutatják a módszer hibáját $h_1 = 0.1$ és $h_2 = 0.01$ lépéstávolság mellett.

t_n	$u(t_n)$	$y(t_n)$	z_n
0	1.0000	1.0000	0
0.1000	1.0048	1.0048	4.0847e-006
0.2000	1.0187	1.0187	7.3920e-006
0.3000	1.0408	1.0408	1.0033e-005
0.4000	1.0703	1.0703	1.2104e-005
0.5000	1.1065	1.1065	1.3690e-005
0.6000	1.1488	1.1488	1.4865e-005
0.7000	1.1966	1.1966	1.5692e-005
0.8000	1.2493	1.2493	1.6227e-005
0.9000	1.3066	1.3066	1.6518e-005
1.0000	1.3679	1.3679	1.6607e-005

t_n	$u(t_n)$	$y(t_n)$	z_n
0	1.0000	1.0000	0
0.0100	1.0000	1.0000	4.1583e-010
0.0200	1.0002	1.0002	3.3825e-008
⋮	⋮	⋮	⋮
0.1000	1.0048	1.0048	1.8521e-006
⋮	⋮	⋮	⋮
0.5000	1.1065	1.1065	1.2194e-005
⋮	⋮	⋮	⋮
0.9000	1.3066	1.3066	1.5515e-005
⋮	⋮	⋮	⋮
0.9900	1.3616	1.3616	1.5233e-005
1.0000	1.3679	1.3679	1.5087e-005

Itt azt is láthatjuk, hogy alig javult a módszer pontossága. Ez ugyancsak azért van, mert az ODE23 is változó lépésközű és elérte a pontosság a beállított default értéket.

	ODE45	ODE45	ODE23	ODE23
h	0.1	0.01	0.1	0.01
z_{n^*}	1.2090e-009	1.0903e-009	1.6607e-005	1.5087e-005

A táblázatból láthatjuk, hogy ennél a példánál az ODE45 pontosabb, mint az ODE23. Az ODE23 algoritmust hatékonyabb lehet nagyobb hibahatárnál vagy, ha differenciálegyenlet enyhén merev.

3. fejezet

Többlépéses módszerek

Az eddig vizsgált módszerek mind egylépésesek voltak, melyeknél t_0 pontból indulva, ismerve y_0 értékét megpróbáltuk meghatározni t_1-et. Ezután t_1 ismeretében közelítettük t_2-t y_1 segítségével, de itt már nem használtuk fel y_0 ismeretét, és így tovább. A többlépéses módszereknél az alapötlet az, hogy az adott pontbeli megoldás ne csak az eggyel előző értéktől függjön, hanem a többi korábbitól is.

Egy általános r-lépéses numerikus módszer általános alakja a következőképpen írható fel:

$$\frac{a_0 y_n + a_1 y_{n-1} + \ldots + a_r y_{n-r}}{h} = b_0 f_n + b_1 f_{n-1} + \ldots + b_r f_{n-r} \tag{3.1}$$

ahol $a_0, \ldots, a_r, b_0, \ldots, b_r$ a módszert leíró paraméterek és f_n $f(t_n, y_n)$-t jelöli. Hogy egy módszer egyértelmű legyen (ún. normalizáló) feltételt adunk meg:

$$\sum_{k=0}^{r} b_k = 1 \tag{3.2}$$

Ha ki tudjuk kifejezni a legmagasabb deriváltat explicit módon, azaz $b_0 = 0$, akkor a módszer explicit, ha nem tudjuk kifejezni, azaz $b_0 \neq 0$, akkor pedig implicit.

3.1. A lineáris többlépéses módszer rendje

A módszer indulásához kellenek az $y_0, \ldots, y_{r-1}$ értékek. Ezek vagy adottak, vagy egy másik módszerrel kell meghatározni őket (például Runge-Kutta módszerekkel).

A (3.1) lineáris többlépéses módszernek is csak a konzisztenciát fogjuk vizsgálni, a stabilitás itt is érvényes.

Hogyan válasszuk meg $a_0, \ldots, a_r, b_0, \ldots, b_r$ $2r+2$ darab együtthatót, hogy a módszer adott p-ed rendű legyen?

Lokális approximációs hiba:

$$\psi_n^{(1)} = -\frac{1}{h}\sum_{k=0}^{r} a_k y(t_{n-k}) + \sum_{k=0}^{r} b_k f(t_{n-k}, y_{t_{n-k}})$$

Fejtsük Taylor-sorba $y(t_{n-k})$ és $f(t_{n-k}, y_{t_{n-k}})$-t úgy, hogy majd a módszer p-ed rendben konzisztens legyen.

$$y(t_{n-k}) = y(t_n - kh) = \sum_{k=0}^{p}(-1)^l \frac{(kh)^l}{l!} y^{(l)}(t_n) + \mathscr{O}(h^{p+1}),$$

$$f(t_{n-k}, y_{t_{n-k}}) = y'(t_{n-k}) = \sum_{k=0}^{p-1}(-1)^l \frac{(kh)^l}{l!} y^{(l+1)}(t_n) + \mathscr{O}(h^{p}).$$

Ha $l = 0$, akkor kapjuk a következő feltételt:

$$\sum_{k=0}^{r} a_k = 0. \tag{3.3}$$

Ha $l = 1, 2, ..., p$, akkor:

$$\sum_{k=0}^{r} a_k (-1)^{l+1} \frac{k^l h^{l-1}}{l!} y^{(l)}(t_n) + \sum_{k=0}^{r} b_k (-1)^{l-1} \frac{(kh)^{l-1}}{(l-1)!} y^{(l)}(t_n) = 0.$$

Osszuk el mindkét oldalt h^{l-1}-gyel, $(-1)^{l-1}$-gyel és $y^{(l)}(t_n)$-nel:

$$\sum_{k=0}^{r} a_k \frac{k^l}{l!} + \sum_{k=0}^{r} b_k \frac{k^{l-1}}{(l-1)!} = 0.$$

Vonjuk össze a szummákat, szorozzuk fel $l!$-sal és emeljük ki k^{l-1}-et:

$$\sum_{k=0}^{r} k^{l-1} \left(k a_k + l b_k\right) = 0. \tag{3.4}$$

Ez p darab újabb feltétel. A (3.2), (3.3), (3.4) egyenletekből $p+2$ darab feltételünk van és $2r+2$ darab paraméterünk, tehát $p \leq 2r$ lehet legfeljebb a módszer rendje.

3.2. Adams típusú módszerek

Bashforth (1883) egy kapilláris jelenség leírására egy matematikai modellt készített. Ennek megoldására javasolta Adams a módszert.

$$y(x_{k+1}) - y(x_k) = \int_{x_k}^{x_{k+1}} f(x, y(x))\, dx.$$

Legyen az egyszerűség kedvéért $f_k = f(x_k, y(x_k))$. Illesszünk interpolációs polinomot az $(x_{k-r}, f_{k-r}), \cdots, (x_k, f_k)$ pontokra és azt integráljuk az adott intervallumon.

$$\int_{x_k}^{x_{k+1}} \sum_{i=0}^{r} f_{k-i} l_{k-i}(x)\, dx$$

$$= \sum_{i=0}^{r} f_{k-i} \int_{x_k}^{x_{k+1}} l_{k-i}(x)\, dx,$$

ahol $\int_{x_k}^{x_{k+1}} l_{k-i}(x)\, dx$ előre kiszámíthatóak. Ebből kapjunk explicit Adams módszert vagy más néven Adams-Bashforth-módszert:

$$\frac{y_n - y_{n-1}}{h} = b_1 f_{n-1} + \ldots + b_r f_{n-r}.$$

Az Adams-Moulton-módszerhez most az $(x_{k-r}, f_{k-r}), \ldots, (x_{k+1}, f_{k+1})$ pontokra illesszünk polinomot és azt integráljuk az $[x_k, x_{k+1}]$ intervallumon.

$$\int_{x_k}^{x_{k+1}} \sum_{i=-1}^{r} f_{k-i} l_{k-i}(x)\, dx$$

$$= \sum_{i=-1}^{r} f_{k-i} \int_{x_k}^{x_{k+1}} l_{k-i}(x)\, dx,$$

ahol $\int_{x_k}^{x_{k+1}} l_{k-i}(x)\, dx$ előre kiszámíthatóak. Az előzőek alapján a következő módszerhez jutunk:

$$\frac{y_n - y_{n-1}}{h} = b_0 f_n + b_1 f_{n-1} + \ldots + b_r f_{n-r},$$

ahol $b_0 \neq 0$. Ez egy implicit séma, f_{k+1}-hez szükségünk van y_{k+1}-re. Az induláshoz kellenek $y_0, \ldots, y_{r-1}$ értékek. Ezek vagy adottak, vagy egy másik módszerrel kell meghatározni őket. Az Adams-Bashforth-módszerrel együtt prediktor-korrektor párként használják, azaz először Adams-Bashforth-módszerrel meghatározzuk u_{k+1} egy y_{k+1} közelítését, majd ennek segítségével az Adams-Moulton-módszerrel meghatározzuk y_{k+1}-et. Összefoglaló néven Adams típusú módszereknek nevezzük azokat a többlépéses módszereket, ahol $a_0 = 1$, $a_1 = -1$, $a_2 = \ldots = a_r = 0$. Általános alakjuk:

$$\frac{y_n - y_{n-1}}{h} = b_0 f_n + b_1 f_{n-1} + \ldots + b_r f_{n-r} \tag{3.5}$$

Ha a módszer explicit, azaz ha $b_0 = 0$, akkor a módszer Adams-Bashforth módszernek nevezzük, míg, ha a módszer implicit, $b_0 \neq 0$, akkor Adams-Moulton módszernek.

3.3. Többlépéses módszer a Matlabban

Természetesen többlépéses módszerek algoritmusai is megtalálhatók a Matlabban. Ilyen például az ODE113, mely rendje 1-től 13-ig változhat és az Adams-Bashforth-Moulton PECE módszeren alapszik. Az algoritmus először egy Adams-Bashforth módszerrel "prediktálja" y_{n+1}-et, majd az Adams-Moulton módszer "korrigálja" azt. A PECE módszert (P = prediction step, E = evaulate, C = correction step) prediktor-korrektor módszernek (magyarul: jósló-javító módszernek) is szokás nevezni. A módszer egy explicit és egy implicit módszer egymás utáni alkalmazása.

- Prediktor: Egy explicit módszer, amellyel megmondjuk, hogy az iterációt honnét indítsuk az implicit módszer esetén.
- Korrektor: Az alkalmazott implicit módszer, amellyel finomítjuk y_{n+1} értékét.

Futtassuk le a példánkra ezt a programot is. Hívjuk meg a következőképpen, $h_1 = 0.1$ lépésközre: `[T1, Y113] = ode113(@diffegy, T1, 1)`, majd $h_2 = 0.01$ lépéstávolságra: `[T2, X113] = ode113(@diffegy, T2, 1)`. A módszer pontosságát az alábbi táblázatokban láthatjuk:

t_n	$u(t_n)$	$y(t_n)$	z_n
0	1.0000	1.0000	0
0.1000	1.0048	1.0048	6.2300e-006
0.2000	1.0187	1.0187	1.8714e-005
0.3000	1.0408	1.0408	2.7885e-005
0.4000	1.0703	1.0703	2.1933e-005
0.5000	1.1065	1.1065	1.8889e-005
0.6000	1.1488	1.1488	1.7254e-005
0.7000	1.1966	1.1966	1.5668e-005
0.8000	1.2493	1.2493	1.4228e-005
0.9000	1.3066	1.3066	1.2872e-005
1.0000	1.3679	1.3679	1.1643e-005

t_n	$u(t_n)$	$y(t_n)$	z_n
0	1.0000	1.0000	0
0.0100	1.0000	1.0001	1.6625e-007
0.0200	1.0002	1.0002	1.4800e-007
⋮	⋮	⋮	⋮
0.1000	1.0048	1.0048	1.4004e-007
⋮	⋮	⋮	⋮
0.5000	1.1065	1.1065	1.7178e-008
⋮	⋮	⋮	⋮
0.8000	1.2493	1.2493	2.0090e-008
⋮	⋮	⋮	⋮
0.9000	1.3066	1.3066	1.8052e-008
⋮	⋮	⋮	⋮
0.9800	1.3553	1.3553	1.6692e-008
0.9900	1.3616	1.3616	1.6525e-008
1.0000	1.3679	1.3679	1.6360e-008

4. fejezet

Összefoglalás

Célunk az volt, hogy numerikus megoldást találjunk közönséges differenciálegyenlet kezdetiérték feladataira. Erre számos módszert néztünk. Megismerkedtünk egylépéses és többlépéses módszerekkel is. Mindegyik módszernek kiszámoltuk a rendjét és egy adott példán teszteltük a pontosságukat. Az alábbi táblázatban összefoglaltam azon módszerek pontosságát $t^* = 1$ pontban, melyekkel a szakdolgozatom foglalkozott:

módszer	z_{n^*}	
	$h_1 = 0.1$	$h_2 = 0.01$
explicit Euler	1.9201e-002	1.8471e-003
javított Euler	6.6154e-004	6.1775e-006
implicit Euler (pontosabb)	1.7664e-002	1.8318e-003
implicit Euler	2.1537e-002	1.8687e-003
ODE45	1.2090e-009	1.0903e-009
ODE23	1.6607e-005	1.5087e-005
ODE113	1.1643e-005	1.6360e-008

Természetesen a Matlabban nemcsak az itt szerepelt beágyazott módszerek léteznek. Ezek és az ODESET függvény bemutatása, a merev rendszerek vizsgálata meghaladja a szakdolgozat kereteit, így csak egy röviden foglaltuk össze ezen beágyazott módszerek főbb jellemzőt.

Az ODE45 egy explicit Runge-Kutta formulán alapszik, méghozzá a Dormand-Prince páron. Általában ez az algoritmus a legjobb első próbálkozásnak a legtöbb differenciálegyenletre.

Az ODE23 is explicit Runge-Kutta módszeren alapszik, a Bogacki-Shampine páron. Hatékonyabb lehet az ODE45-nél, ha nagyobb a hibahatár vagy, ha differenciálegyenlet enyhén merev.

Az ODE113 változó rendű Adams-Bashforth-Moulton PECE módszer. Hatékonyabb lehet az ODE45-nél, ha szigorúbb hibahatárnál vagy, ha a differenciálegyenletet drága kiszámolni.

Ezeket a módszereket nem merev feladatok megoldására tervezték, így ha nagyon lassúak, akkor inkább egy másikat próbáljunk az alábbiak közül.

Az ODE15S változó rendű módszer, mely a differencia képleteken alapszik. Általában a Gear-módszert vagy más néven BDF módszert használja, ami kevésbé hatékony. Az ODE15S is többlépéses, mint az ODE113. Használjuk ezt, ha az ODE45 elbukik, vagy nem hatékony, ha a feladat merev, vagy algebrai differenciálegyenlet-rendszert akarunk megoldani.

Az ODE23S egy módosított másodrendű Rosenbrock formulán alapszik. Ez egy egylépéses módszer, így eredményesebb lehet az ODE15S-nél nagyobb hibahatárnál. Több olyan merev rendszert is meg tud oldani, amelyeknél az ODE15S nem hatékony.

Az ODE23T a trapéz szabály egyfajta megvalósítása. Használjuk ezt a megoldót, ha a feladat csak enyhén merev és, ha a megoldást számítási hibák nélkül szeretnénk megkapni. Az ODE23T algebrai differenciálegyenlet-rendszert is meg tud oldani.

Az ODE23TB a TR-BDF2 módszer megvalósítása, egy implicit Runge-Kutta módszer, melynek első lépése egy trapéz szabály, a második lépés pedig egy másodrendű differencia képlet. A konstrukció miatt mindkét lépésben ugyanazt az iterációs mátrixot használja. Hasonlóan, mint az ODE23S hatékonyabb lehet az ODE15S-nél nagy hibahatárnál.

Végül pedig az ODE15I változó lépésközű algoritmus. Ez a módszer teljesen implicit differenciálegyenletek megoldására alkalmas.

Köszönetnyilvánítás

Köszönettel tartozom témavezetőmnek, Faragó Istvánnak, odaadó segítségéért és türelméért. Köszönöm Valkó Évának és Karsai Tamásnak az ötleteiket és tanácsaikat, melyek segítségével munkám eredményesebb lehetett. Hálás vagyok családomnak és a szeretteimnek támogatásukért.

Irodalomjegyzék

[1] Bogacki, P. and L. F. Shampine, A 3(2) pair of Runge-Kutta formulas, Appl. Math. Letters, Vol. 2, 1989.

[2] Dormand, J. R. and P. J. Prince, *A family of embedded Runge-Kutta formulae*, J. Comp. Appl. Math., Vol. 6, 1980.

[3] Shampine, L. F. and M. K. Gordon, *Computer Solution of Ordinary Differential Equations: the Initial Value Problem*, W. H. Freeman, SanFrancisco, 1975.

[4] J. David Logan, *A First Course in Differential Equations*, Springer Science+Business Media, Inc., 2006.

[5] Internetes forrás, David Houcque, *Applications of MATLAB: Ordinary Differential Equations (ODE)*
`http://www.mccormick.northwestern.edu/docs/efirst/ode.pdf`

[6] Faragó István, Horváth Róbert, *Numerikus Módszerek Egyetemi Jegyzet*, 2011.

[7] Stoyan Gisbert, Takó Galina, *Numerikus Módszerek II*, Typotex, 1995.