EÖTVÖS LORÁND TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

REKURZIÓK SZÁMÍTÓGÉPES PROGRAMOK SEGÍTSÉGÉVEL

-SZAKDOLGOZAT-

Készítette: Csapó Zsuzsanna

(Matematika Bsc, Tanár)

Témavezető: Gémes Margit

(Analízis Tanszék, Matematikai Intézet)

Budapest
2011

Tartalomjegyzék

Oroszlánfogás a Szaharában[1]

Hogyan fog oroszlánt a sivatagban egy matematikus, egy fizikus és egy biológus?
A fizikus készít egy 3×3 méteres szitát félméteres lyukakkal, majd átszitálja a Szaharát. Végül a szitában ott marad az oroszlán.
Mit tesz egy biológus? Megjegyzi halkan, hogy a Szaharában nincs is oroszlán.
És végül a matematikus? Ez később kiderül a szakdolgozatomban.

[1] Fried Katalin, Simonovits Miklós: A problémamegoldás számítógépes iskolája

Bevezetés

Miért pont a rekurzió?

Matematika-informatika tanár szakos hallgatóként, olyan témát szerettem volna választani, ami mindkét tudományágat érinti. Mivel a matematika az alapszakom, így ez a terület lesz a meghatározó. Mai, modern információs társadalmunkban fontos szerepet kap a számítógép használata. Ma már viszonylag kevés olyan gyermek illetve felnőtt éli a mindennapjait, aki ne használná nap mint nap a számítógép nyújtotta lehetőségeket. Ez főleg az internetezésben merül ki: közösségi oldalak, játékok, chat valamint a világban való tájékozódásként is segítségünkre szolgálhat: hírolvasások, kutatások.

A szakdolgozatomban megmutatom, hogy még miért is hasznos ez a mindennapi boldogulást megkönnyítő eszköz. Célom rávezetni az olvasót, hogy kevés számítástechnikai tudással hogyan lehet hatékonyabbá tenni a tanulást. Szeretnék rámutatni arra, hogy hogyan használható a számítógép az analízisbeli fogalmak szemléltetésére, illetve az analízis feladatok megoldásának a megsejtésére. Nézünk könnyebb, illetve olyan matematikai feladatokat is, melyek ránézésre talán nem triviálisak, de egy egyszerű, pár soros program segítségével, könnyebbé válhat a feladat megértése, majd a feladat megoldása. Mindezt elsősorban a matematikai oldalról, másodsorban a számítástechnika szempontjából vizsgáljuk meg.

Több programnyelvre is lesz példa a dolgozatban. A PASCAL nyelv választásakor több szempontot vettem figyelembe: először is olyan nyelvet kellett keresnem, amelyik mindenki számára elérhető (FREE PASCAL v2.24). Másodszor pedig mivel ez a nyelv tipikusan az oktatásra lett fejlesztve, így célszerű ebben elkezdeni a tanulást, gyakorlást. Ha itt megértjük a programozás elvét, akkor a többi nyelvvel sem lesz problémánk a későbbiekben. A programok hatékonyságával (gyorsaság, helyfoglalás) most itt nem foglalkozunk, a fő cél az átláthatóság lesz. A kicsik körében közkedvelt a Logó nyelv. Erre is mutatok példát. Véleményem szerint a rekurzió megértése Logóban egyszerű, hiszen a megértést segítheti a teknős által kirajzolt ábra is (Comenius Logo 3.0). Végül a fizikai modellezési feladat QBASIC-ben (qb 1.0) lesz megírva. Ez a nyelv is ingyenes. Azért térek át erre a környezetre, mert a feladat célja a fizikai mozgás pályájának kirajzolására lesz, aminek megvalósítása ebben a környezetben egyszerűbben megoldható. És nem utolsó sorban így össze lehet vetni a PASCAL-t a QBASIC-kel, annak érdekében, hogy felfedezhessük a két nyelv hasonlóságait és különbségeit.

Munkám során több középiskolás matematika könyvet lapoztam át, több gyakorlattal rendelkező tanárt kérdeztem meg, és azzal szembesültem, hogy a középiskolás törzsanyag nem tartalmazza a rekurzív sorozatok elméletét. Egy általános tantervű osztályban a számtani és mértani sorozatokon kívül más sorozat nem kerül szóba. A rekurziót, mint fogalmat a gyerekek nem ismerik meg. Az emelt szintű érettségire készülő osztályban a témakör kibővül a teljes indukciós bizonyítással és a Fibonacci-sorozattal, esetleg a határérték számítással, de az már nagyon ritka. Mégsem mindig tudják hova tenni a tanulók a rekurzió fogalmát az elmélet megalapozásának a hiánya miatt. Tapasztalatom, hogy elhanyagolják a rekurzív gondolkodásmód fejlesztését a középiskolákban. A rekurziók világa pedig nagyon izgalmas téma lehet a gyerekeknek, hiszen a megszokottól eltérő gondolkodásmódot követel meg a feladatok megoldása során.

A rekurziós problémák a számítógépek elterjedése óta még inkább előtérbe kerültek, hiszen megtehetjük, hogy kiszámoljuk a sorozat első néhány tagját, majd megpróbáljuk kideríteni, hogy a továbbiakban hogyan viselkedhet a sorozatunk. A számítógép gyorsan, nagyon sok lépést tud végrehajtani, így egy rekurzívan megadott sorozat tagjainak a felsorolása innentől kezdve gyerekjáték lesz. A rekurziók fontosságát mutatja továbbá, hogy a mindennapi életünkben a jelenlegi helyzetből nemcsak a közeli (tehát az n-ből az $(n+1)$-re), hanem a távoli jövőre is (közvetlenül a k-ra, ahol k akármennyi lehet) következtethetünk.

Dolgozatom első részében a rekurzív sorozatok elméletét tárgyalom, néhány egyszerűbb középiskolásoknak szóló feladattal bemutatva. A fejezet végén pedig az írásom elején feltett humoros kérdésre adom meg a választ. Utána a Newton-módszert mutatom be pár nehezebb, egyetemi szintű feladattal, amelyek megkívánhatják az összetettebb gondolkodásmódot, mind matematika, mind az informatika terén. Végül pedig egy szimulációs modell, egy fizikai alkalmazás következik, ahol majd láthatjuk, hogy milyen párhuzamot húzhatunk a differenciaegyenletek és a rekurziók között.

Remélem, hogy sok hasznos dolgot talál munkámban az Olvasó és esetleg meghozom a kedvét a programjaim tovább gondolásához, fejlesztéséhez. Legfőbb célom ez: használjuk az eszünket és a gépünket!

Elméleti bevezető

A matematikában a halmazt alapfogalomnak tekintjük, amelyet más fogalmakkal nem definiálunk. A halmazt alkotó dolgokat a halmaz elemeinek nevezzük, amelyen valamely dolgok összességét értjük. Halmazok között vizsgálhatunk különböző kapcsolatokat, relációkat. A relációk alapján egy halmaz elemeihez hozzárendelhetünk elemeket egy másik halmazból. A hozzárendelés maga definiálatlan fogalom. Egyértelmű a hozzárendelés, ha az alaphalmaz bármelyik eleméhez a képhalmaz legfeljebb egy elemét rendeljük hozzá.

Valamely A és B nem üres halmazok közötti δ relációt függvénynek nevezzük, ha az A halmaz minden egyes eleme a B halmaznak egy és csak egy elemével van relációban. Tehát:

- ha $a \in A$, akkor van olyan $b \in B$, hogy $(a, b) \in \delta$,
- ha $a \in A$; $b_1 \in B$; $b_2 \in B$, továbbá $(a, b_1) \in \delta$ és $(a, b_2) \in \delta$, akkor $b_1 = b_2$.

Számsorozaton olyan függvényt értünk, amelynek értelmezési tartománya a pozitív egész számok halmaza, az értékkészlete pedig a valós számok egy részhalmaza. A számsorozatok leírására az a_1,a_2,a_3,…a_n,…az indexes jelölést használjuk, ahol az a_n a sorozat n-edik tagját jelöli.

A sorozat explicit megadása azt jelenti, hogy az n-edik tagot egy, csak az n-től függő képlettel adjuk meg. Ezzel szemben a rekurzív formula olyan egyértelmű utasítás, amikor a sorozat tagjait a korábbi tagok segítségével fejezhetjük ki. Ekkor azonban előre meg kell adni bizonyos számú kezdőtagot, hiszen így tudjuk a rákövetkezőeket meghatározni. Azért lehet szükség az explicit alakra, mert segítségével a sorozat könnyen kezelhetővé válik. Tehát egy rekurzív sorozat megoldásán, az explicit alak meghatározását értjük.

Elméleti anyagként még meg kell említeni az explicit alak meghatározására a teljes indukció közti kapcsolatot is. Általános eljárás, hogy a rekurzív összefüggés alapján felírjuk az első egynéhány tagját a sorozatnak. Ezekkel az adatokkal már sejthetjük, hogy mi lesz az explicit alak, majd ezt teljes indukcióval bebizonyítjuk. Az elv a következő: Adott egy n pozitív egész számtól függő állítás. Először bebizonyítjuk az állítást $n = 1$-re, majd feltesszük, hogy az állítás igaz n-re (ezt nevezzük indukciós feltevésnek), és bebizonyítjuk, hogy igaz $n + 1$-re is. Ezzel az állítást minden n pozitív egész számra bebizonyítottuk.

Középiskolában, mint már említettem a kötelező tananyag a számtani és mértani sorozatok elsajátítása. A definiálásuk rekurzív módon is történhet, kiszámításuk pedig az explicit képlet segítségével, anélkül, hogy ismernék a rekurzió fogalmát. Tehát ha a rekurziókat megpróbálnánk osztályozni, akkor először is meg kell említenünk e két fontos sorozatot.

A rekurziók egy lehetséges osztályozása:[3]

1. Állandó együtthatós, elsőrendű, $d = 0$ esetén homogén, egyébként inhomogén, lineáris rekurzió: $a_n = q \cdot a_{n-1} + d$ $(n \geq 2)$.

   Speciális esetei:

   - számtani sorozat: $a_1 = c,\ a_n = a_{n-1} + d$ $(c, d$ állandó), melynek explicit formulája: $a_n = c + (n-1) \cdot d$, $(n \geq 1)$.
   - mértani sorozat: $b_1 = c,\ a_n = q \cdot a_{n-1}$ $(c, q \neq 0$ állandók), melynek explicit formulája: $a_n = c \cdot q^{n-1}$, $(n \geq 1)$.

2. Nem állandó együtthatós, elsőrendű, homogén, lineáris rekurzió:

   $a_n = f(n) \cdot a_{n-1}$ $(n \geq 1)$.

   Példa: $c_0 = 1$, $a_n = n \cdot a_{n-1}$, melynek megoldása $a_n = n!$ $(n \geq 0)$.

3. Állandó együtthatós, másodrendű, homogén, lineáris rekurzió.

   Például: $f_0 = 0,\ f_1 = 1,\ f_n = f_{n-1} + f_{n-2}$ $(n \geq 2)$. Ezzel a középiskolában is találkozhatunk: a Fibonacci-sorozat. Megoldása bonyolult, középiskolában nem kerül elő: $a_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}^n}{2}\right)\right]$.

Sok esetben egy sorozat tagjai az $a_{n+k} = c_1 a_{n+k-1} + \cdots + c_k a_n$ (k rögzített, $c_1, .. c_k \in \mathbb{R}$) alakú rekurziónak tesznek eleget. Az ilyen rekurziót állandó együtthatós lineáris rekurziónak nevezzük. Az állandó együtthatós jelző arra utal, hogy $c_1, \ldots c_k$ együtthatók rögzítettek, tehát nem függnek n-től, míg a lineáris tulajdonság arra, hogy az a_{n+k}-t megadó jobboldali kifejezés az $a_{n+k-1}, \ldots, a_n$ lineáris kifejezése. A rekurzív sorozatot tehát többféleképpen jellemezhetjük. Ha nem szerepel benne konstans tag, akkor homogén, egyébként pedig inhomogén rekurzióról beszélhetünk. A rendűség jelző pedig nem más, mint, hogy egy rekurzív sorozatnak hány korábbi tagja szerepel.

Vizsgáljuk meg részletesebben a $k = 2$ esetet, vagyis az $a_{n+2} = c_1 a_{n+1} + c_2 a_n$ alakú rekurziót. A megoldását mértani sorozat formájában keressük, tehát legyen $a_n = a_1 q^{n-1}$. Ezt behelyettesítve a rekurzióba a $a_1 q^{n+1} = c_1 a_1 q^n + c_2 a_1 q^{n-1}$-t kapjuk. Most $a_1 q^{n-1}$-nel leosztva egy másodfokú egyenlethez jutunk: $q^2 - c_1 q - c_2 = 0$-hoz. Ezt nevezzük a lineáris rekurzió karakterisztikus egyenletének, gyökeit pedig jelöljük q_1-gyel és q_2-vel. a_1-et tetszőlegesen választva az $a_n = a_1 \cdot {q_1}^{n-1}$és $a_n = a_1 \cdot {q_2}^{n-1}$ mértani sorozatok kielégítik a kiindulási rekurziót, sőt a linearitás miatt az $a_n = \lambda_1 \cdot {q_1}^{n-1} + \lambda_2 \cdot {q_2}^{n-1}$ alakú sorozatok is eleget tesznek, ahol λ_1, λ_2 tetszőleges rögzített számok.

Ha az egyenletnek két különböző gyöke van, az már elegendő a kiindulási rekurzió megoldásához, ugyanis λ_1, λ_2 együtthatókat megválaszthatjuk úgy, hogy a sorozat első két tagja a_1 és a_2 legyen. Ekkor az alábbi lineáris egyenletrendszernek mindig lesz egyértelmű megoldása.

$$\left.\begin{aligned} a_1 &= \lambda_1 + \lambda_2 \\ a_2 &= \lambda_1 \cdot q_1 + \lambda_2 \cdot q_2 \end{aligned}\right\}$$

Az is előfordulhat, hogy a karakterisztikus egyenletnek komplex gyökei vannak, vagyis a fenti egyenletrendszert a komplex számok halmazán kell megoldani. Mivel az együtthatók valós számok, így ezek a komplex gyökök egymás konjugáltjai.

Olyan is előfordulhat, hogy $q_1 = q_2$ kétszeres gyök. Ekkor olyan egyenlethez jutunk, aminek q^{n-1}-en kívül $n \cdot q^{n-1}$ is eleget tesz a rekurziónak. Némi számolás után a $2 \cdot q^2 = c_1 \cdot q$ egyenletet kapjuk, ami nem más, mint a karakterisztikus egyenlet deriváltja. Ezzel beláttuk, hogy $n \cdot q^{n-1}$ is gyöke lesz a rekurziónak, hiszen egy többszörös gyök a polinom deriváltjának is gyöke.

Tetszőleges k esetén teljesen analóg módon járhatunk el, azonban a számítások bonyolultabbá válhatnak és nehézséget okozhat a karakterisztikus egyenlet gyökeinek a megkeresése. Ekkor a karakterisztikus egyenlet k-ad fokú lesz, gyökei pedig legyenek q_1, q_2, …, q_k. Abban az esetben, ha $q_1 = q$ r-szeres gyök, akkor q^{n-1}, $n \cdot q^{n-1}$, … ,$n^{r-1} \cdot q^{n-1}$ is eleget tesz a rekurziónak, vagyis egy r-szeres gyök esetén a felírt tagok kerülnek az egyenletrendszerbe, ami szintén egyértelműen megoldható lesz. [4]

Léteznek egyéb rekurziók is, ami azonban már nem tárgya a szakdolgozatomnak, mégis érdemes a megemlítés szintjén pár szót ejteni róluk. A nem lineáris rekurziók lényegesen nehezebbek. Nincs általános eljárás, minden feladat specialitásától függ a megoldás. Szimultán rekurziók esetében pedig több, egymásra kölcsönösen hivatkozó sorozat rekurzív alakja adott.[3]

Nézzünk pár egyszerűbb feladatot!

Feladatok

1. Elsőrendű rekurzió

1.1. Feladat:[2] Keressük meg az $a_n = x_n \cdot a_{n-1}$ $(n \geq 1)$ elsőrendű, lineáris, homogén rekurziós képlet megoldását!

Helyettesítsünk be $n = 1,2,3$ ...-at, és figyeljük az eredményeket: $a_1 = x_1 \cdot a_0$, $a_2 = x_2 \cdot a_1 = x_2 \cdot x_1 \cdot a_0$, $a_3 = x_3 \cdot a_2 = x_3 \cdot x_2 \cdot x_1 \cdot a_0$. Ha ezt folyatatjuk, akkor az $a_n = x_n \cdot x_{n-1} \cdot \ldots \cdot x_2 \cdot x_1 \cdot a_0$ általános alakot kapjuk.

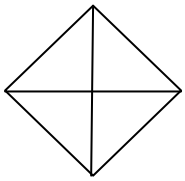
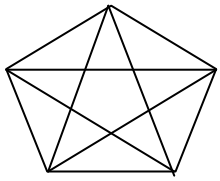
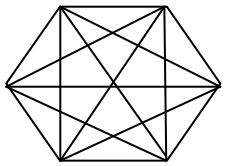
Vizsgáljuk meg!

- Ha $a_0 = 0$, akkor $a_n = 0$ $\forall$ n-re. Ez a triviális megoldás.
- Ha $x_n = x$ n-től függetlenül, akkor $a_n = x^n \cdot a_0$. Ebben az esetben az x^n hatványok e rekurziós képlet egyértelműen meghatározott megoldásai az $x^0 = 1$ kezdeti feltétel mellett.
- Ha $x_n = n$ $\forall n$-re, akkor az $a_0 = 1$ kezdeti feltétel mellett az $a_n = n!$ sorozatot kapjuk. Explicit alakja tehát $n! = n \cdot (n-1) \cdot ... \cdot 3 \cdot 2 \cdot 1$.

1.2. Feladat: Hány átlója van egy konvex n-szögnek?

Először nem rekurzívan nézzük meg. Így ez már általános iskolában is megoldható. Tudjuk, hogy minden csúcsból $n-3$ átlót húzhatunk. Akkor ez n-szög esetében $n \cdot (n-3)$ átlót jelent. Mivel minden átlót kétszer számoltunk, az összes átlók száma: $\frac{n \cdot (n-3)}{2}$.

Nézzünk példákat:

négyszög	ötszög	hatszög
		
$A_4 = \frac{4 \cdot (4-3)}{2} = 2$	$A_5 = \frac{5 \cdot (5-3)}{2} = 5$	$A_6 = \frac{6 \cdot (6-3)}{2} = 9$

Nézzük meg egy egyszerű PASCAL nyelvű példaprogramot:

```
begin
  clrscr;
  writeln('Hany atloja van egy konvex n-szognek?');
  for n:=3 to 12 do
   begin
    A:=(n*(n-3)/2);
    writeln(n,A:5:0);
  end;
readkey;
end.
```

A dolgozatomban helyszűke miatt csak programrészletek szerepelnek. A teljes programok megtalálhatóak a mellékelt CD-n.

Ez a viszonylag egyszerű program megfelelő a programozás elkezdéséhez. A `CLRSCR` egy beépített függvény a `CRT` unitban a képernyőtörlés felelőse. A unit egy olyan önállóan lefordítható programegység, ahol az egyes unitok újabb eljárásokat, függvényeket, konstansokat és típusokat tartalmaznak. A deklarációs részben állandókat, változókat és saját típusokat készíthetünk. Saját típusokat a `TYPE`, állandókat a `CONST`, változókat `VAR` kulcsszó után lehet deklarálni. Mind az

eljárásnak, mind a függvénynek adhatunk át paramétert, amit csak az adott eljárás vagy függvény fog látni (lokális változó lesz), ellenben a függvény vissza is ad egy értéket. Az eljárások és a függvények korlátlanul egymásba ágyazhatók. Az eljárást a PROCEDURE eljárásnév (esetleges paraméter); kulcsszóval, a függvényt a FUNCTION függvénynév (esetleges paraméter): visszaadott érték típusa. Egy egyszerű PASCAL program felépítése:

```
  program Hello_World;
{használt unitok megadása}
  uses crt,dos;
{deklarációs rész: típusok, konstansok, változók}
  const Konstans = 42;
  type szabadnap = (szombat,vasarnap);
  var Valtozo: integer;
          nap: szabadnap;
{eljárások és függvények}
 procedure Eljaras(Parameter: boolean);
        begin
        end;
 function Fuggveny(Parameter: real): integer;
       begin
    Fuggveny := round(Parameter)
       end;
{programtörzs}
 begin
 end.
```

A WRITELN utasítással kiírathatunk a képernyőre, míg a READLN-nel (habár ebben a programban nem szerepel, mégis szorosan összetartozik a writeln-nel) adatokat kérhetünk be a billentyűzetről. Növekményes vagy számlálós ciklus esetén (FOR) a ciklusmagot (itt: `A:=(n*(n-3)/2); writeln(n,A:5:0);`) egy előre meghatározott számszor (`n:=3-tól 12-ig`) hajtjuk végre, majd értékadással meghatározzuk az átlók számát (A), és kiíratjuk a képernyőre az n-eket és az A-kat. Az `A:4:0` azt jelenti, hogy milyen hosszan kívánunk megadni egy numerikus változót. Jelen esetben a 4 hosszúságot kapta, tehát a képernyő balszélétől számított 4. leütésnél ér véget a szám. Az A változó típusa valós (`real`) lett, hiszen ez a típus nagyobb számok tárolására is alkalmas, mint az egész, ráadásul tizedeseket is képes kezelni. (Itt az osztás művelet miatt volt rá szükség). A jelölésben `a:0` csak annyit jelent jelen esetben, hogy egyetlen számot sem szeretnénk kiírni a tizedesvessző után. A READKEY szintén egy beépített függvény, értéke az a karakter lesz, amelyik billentyűt lenyomjuk. Itt az Enter leütésével kilépünk a programból.

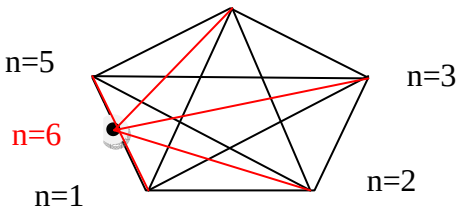
Vegyünk egy segédtáblázatot:

3	4	5	6	7	…	(n-1)	n
0	2	5	9	14	…	$\frac{(n-1)\cdot(n-4)}{2}$	$\frac{n\cdot(n-3)}{2}$

2 3 4 5 n-2

Tehát a sejtésünk: $A_n = 2 + 3 + 4 + \cdots + (n-2)$.

Most oldjuk meg rekurzív gondolatmenettel. Ez már középiskolás feladatnak minősül, minimális technikai tudást igényel. Tegyük fel, hogy a konvex $(n-1)$-szög átlóinak a száma A_{n-1}. Most sorszámozzuk be a csúcsokat pozitív körüljárás szerint 1-től $(n-1)$-ig. Az n. pontot vegyük fel az első és $(n-1)$. között. Az n. pontból $n-2$ új átlót tudunk húzni. (Kimarad a két szomszédos csúcs); valamint az 1. és $(n-1)$. csúcs között is keletkezik egy új átló. Például: $n = 4$



Ötszög átlóinak a száma: 5. Ha felvesszük a 6. pontot $6 - 2 = 4$ új átló keletkezik. $5 + 4 = 9$. Tehát egy hatszögnek 9 átlója van.

Vagyis $A_n = A_{n-1} + n - 2$ $(n \geq 4; A_3 = 0)$.

Tehát a megoldás $A_4 = 2$, $A_n = 2 + 3 + 4 + \cdots + (n-2) = \frac{n\cdot(n-3)}{2}$, ha $n > 4$.

2. Másodrendű rekurzió

2.1. Feladat:[4] Keressük meg a Fibonacci-sorozat explicit képletét, vagyis oldjuk meg az $f_0 = 0,\ f_1 = 1,\ f_n = f_{n-1} + f_{n-2}$, $(n \geq 2)$ lineáris rekurziót!

A karakterisztikus egyenlet $q^2 - q - 1 = 0$, melynek gyökei $q_{1,2} = \frac{1\pm\sqrt{5}}{2}$. Így f_n-et $\lambda_1 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^{n-1} + \lambda_2 \cdot \left(\frac{1-\sqrt{5}}{2}\right)^{n-1}$ alakban keressük, az egyenletrendszer pedig két egyenletből áll:

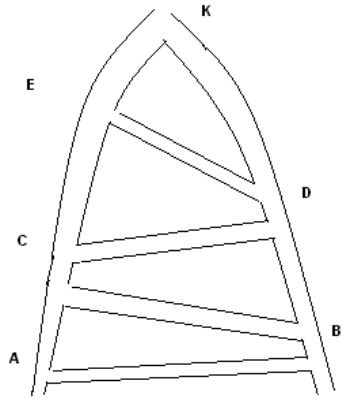
$$\left.\begin{array}{c} (f_0 =)0 = \lambda_1 + \lambda_2 \\ (f_1 =)1 = \lambda_1 \cdot \frac{1+\sqrt{5}}{2} + \lambda_2 \cdot \frac{1-\sqrt{5}}{2} \end{array}\right\}$$

Ennek láthatóan a $\lambda_1 = \frac{1}{\sqrt{5}}$ és $\lambda_2 = -\frac{1}{\sqrt{5}}$ megoldása, amit visszaríva f_n kifejezésbe kapjuk, hogy

$$f_n = \frac{1}{\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n.$$

Ezzel megkaptuk a Fibonacci-sorozat explicit képletét.

2.2. Feladat: Ábránkon két meredek út, és egy azok között kígyózó szerpentin látható. Hányféleképp lehet A-ból eljutni a kilátóhoz (K), ha a meredek utakon és a szerpentinen –akár felváltva is – haladhatunk, de mindig csak felfelé?



Nézzük meg az ábrát! A kiinduló pont az A pont. Ide egyféleképpen juthatunk: úgy, ha helyben maradunk. A B pontba egyféleképpen juthatunk el, a szerpentinen A-ból. A C-be kétféleképpen: vagy a meredek úton A-ból, vagy szerpentinen B-ből. A D pontba vagy C vagy B felől érkezhetünk; a C-ből 2, a B felől 1 lehetőséget jelent, így összesen $2 + 1 = 3$. E-be C-ből 2 lehetőség, vagy D-ből 3 lehetőség van.

Így $2 + 3 = 5$ lehetőség. Megfigyelhető, hogy minden pontba az előző két pont valamelyikéből juthatunk el, így a lehetőségek száma minden pontnál annyi, mint az előző két pontnál összesen. Tehát a Kilátóhoz $3 + 5 = 8$ féleképpen juthatunk fel.

Ehhez egy olyan rekurzív példaprogramot képzelek el, ami az összes Fibonacci-számra visszavezethető feladatnak kiszámítja a megoldását.

```
function fib(n:byte):integer;
begin
   if (n=1) or (n=2) then fib:=1
                     else fib:=fib(n-1)+fib(n-2);
end;
```

Ez egy olyan függvény, ami rekurzívan kiszámolja a bekért n-edik Fibonacci számot. A rekurzív feladatok többsége sok esetben lassítja a programot, és igaz, hogy iteráció (ciklus) segítségével is megoldható lenne, így azonban a végén sokkal

elegánsabb programot kapunk. Egy függvényt vagy eljárást rekurzívnak [8] nevezünk, ha az meghívja saját magát.

Tudjuk, hogy a Fibonacci sorozat első két tagjának az értéke 1. Ez látható az elágazás első sorában. Elágazásokat, szelekciókat (`IF` utasítás) azért szervezünk, hogy megkülönböztessük a különböző eseteket. Az `IF … THEN` jelentése: ha… akkor (egyágú szelekció). A fenti példában az $n = 1$ vagy $n = 2$ illetve az $n > 2$ (`ELSE` ág, többágú szelekció, jelentése különben) eseteket majd a számokat a megelőző kettő összegéből számítjuk ki rekurzívan. Az *n* változó típusa byte, melynek értelmezési tartománya a 0 és 255 közötti egész számok. A főprogramunkban bekérjük a megfelelő Fibonacci-számot, majd meghívjuk a függvényt és kiírattatjuk a megfelelő értéket.

```
begin
  clrscr;
  write('Hanyadik Fibonacci-szamra vagy kivancsi?');
  readln(n);
  write('A keresett szam:');
  writeln(fib(n));
  readkey;
end.
```

Rekurziók intervallumokra [1]

Szakdolgozatom egy tudós viccel nyitottam: hogyan fog oroszlánt a sivatagban egy matematikus, egy fizikus és egy biológus? Utóbbi kettőre már megkaptuk a választ. Most nézzük, hogy mit tesz egy matematikus. Két esetet különböztet meg:

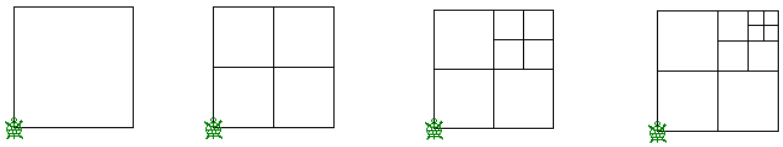
- Az oroszlán egy helyben van;
- Az oroszlán mozog.

Vizsgáljuk meg először az első esetet. Csináljunk egy akkora alul nyitott ketrecet, amiben elfér az oroszlán. Gondolatban tegyük bele az állatot! Felezzük meg a Szaharát Kelet-Nyugat irányában! Ekkor az oroszlán vagy az Északi, vagy a Déli részbe esik. Az is előfordulhat, hogy a határvonalon fekszik, ilyenkor állapodjunk meg, hogy a Déli feléhez soroljuk. A következő lépés, hogy ismét megfelezzük azt a részt ahol az oroszlán található, de most Északi-Déli irányban. Így két sivatag negyedet kapunk és az egyikben van az oroszlánunk.

Most megint felezzük meg ezt Kelet-Nyugat irányában! És így tovább, addig folyatatjuk ezt az algoritmust, amíg olyan kis sivatagrészhez nem jutunk, amely már elfér a ketrec alatt. Így megvan az oroszlán, ha most rátesszük a ketrecet. Miért lehetséges ez? Archimédesz-axiómája szerint: bármely $x \in \mathbb{R}$ számhoz van olyan $n \in \mathbb{N}$, hogy $x < n$. Tehát ez a fontos axióma biztosítja, hogy véges sok lépésben olyan kis sivatagrészhez jutunk, amely már elfér a ketrec alatt. Ezt a módszert felezéses eljárásnak nevezzük, ami mind a matematikában, mind a számítástechnikában egy nagyon hatékony algoritmus. Fontos lépése az eljárásnak, hogy felváltva Kelet-Nyugati, majd Észak-Déli irányban feleztünk. Miért? Ha mindig Kelet-Nyugat irányban csináltuk volna, akkor hosszú, vékony sivatagrészekhez jutnánk. Ami pedig nem lett volna túl szerencsés ebben az esetben. Végül mit tesz a matematikus, ha az oroszlán mozog? Akkor megállapítja, hogy ez az algoritmus nem alkalmazható.

Első látásra talán nem világos, hogy mi köze van a feladatnak a rekurzióhoz. A feladatot mindig visszavezetjük egy előző feladatra, melynek megoldása egy fokkal egyszerűbb. Véges lépésben elérjük a legegyszerűbb esetet, és így megvan a feladat megoldása. A rekurzió egyik alkalmazási területe a fraktálok. Jellemzőjük az önhasonlóság, ami azt jelenti, hogy ha egy-egy ábrát különböző nagyításokban vizsgálunk, újra meg újra ugyanazokra az alapmotívumokra bukkanunk. Nézzünk egy négyszöget rajzoló fraktált [8]:

1. szint 2. szint 3.szint 4.szint



```
tanuld négyzetfraktál :szint :hossz
 ha :szint = 1 [négyzet :hossz]
 ha :szint > 1 [előre :hossz / 2 jobbra 90 előre :hossz / 2 balra 90 ~
                       ismétlés 4 [négyzet :hossz / 2 balra 90] ~
                       jobbra 90 hátra :hossz / 2 balra 90 ~
                        hátra :hossz / 2]
 ha :szint > 2 [előre :hossz / 2 jobbra 90 előre :hossz / 2 balra 90 ~
                       négyzetfraktál :szint - 1 :hossz / 2 ~
                       jobbra 90 hátra :hossz / 2 balra 90 ~
                        hátra :hossz / 2]
vége
```

Ehhez még a négyzet eljárást kell ismernünk.

```
tanuld négyzet :hossz
  Ismétlés 4 [előre:hossz jobbra 90]
vége
```

Ez a program Logó nyelven íródott. Sokan már általános iskolában találkozhatnak ezzel az egyszerű struktúrájú, magyar nyelvű programmal. Hamar elnyerheti a gyerekek tetszését és

meghozhatja a kedvüket a programozáshoz a kedves teknős. Tökéletes ez a nyelv a programozás elkezdéséhez viszonylag kis korban is. Nézzük röviden, hogy működik a program! A négyzet eljárásnak egy paramétere van, az oldal hossza. Négyszer ismételje egymásután, hogy előre megy egy megadott, bekért hosszat, majd jobbra fordul 90˚-ot. Így kirajzol egy négyzetet.

Nézzük a főprogramot. Látjuk, hogy elágazásokra bomlik. Paraméterei a szint és a hossz. Első szint, amikor meghívja a négyzetet rajzoló eljárást. Fontos, hogy ezek állapotátlátszó eljárások, tehát a teknős mindig a kiinduló állapotba ér vissza és felfelé néz. Második szint, amikor már megvan a négyzet, majd elmegy a négyzet közepére és megint meghívja a négyzet eljárást rekurzívan. Láthatjuk az ábrán, hogy egy nagy négyzetből így 4 kicsi lesz. Utána az ötlet ugyanaz: beállunk a jobb felső kis négyzet belsejébe és meghívjuk önmagát rekurzívan, csak szint-1-el. Azaz, ha a 3. szinten tartunk, akkor meghívjuk a 2. szintet, hogy ugyanazt hajtsa végig. És így tovább az előre megadott szintig működik a folyamat.

Láthattuk, hogy a feladatot visszavezettük a legegyszerűbb esetig. A négyszögeket rajzoló algoritmus és a Szahara felezgetése között hasonlóságot vélünk felfedezni. Több alkalmazást is találunk, ami ezt az elvet követi. Ilyen a „gondoltam egy számot, találd ki!” nevezetű játék is. A program minden lépésben megfelezi a keresendő számot tartalmazó intervallumot. Hasznos lehet még a számítástechnikában is, ahol szintén intervallumfelezgetéssel közelíthetjük meg a függvények gyökeit.

Newton-módszer

Bevezető [5]

A Newton-módszer valós, folytonos és differenciálható függvények zérushelyeinek megsejtésére használható iteratív eljárás. Az algoritmus segítségével megkereshetjük a függvények minimumát/maximumát is, azonban ehhez szükséges, hogy létezzen a függvénynek a második deriváltja, ugyanis az első derivált zérushelyei azonosítják a függvény szélsőértékét.

Ezt a módszert először Isaac Newton írta le 1669-ben. Ez a leírás azonban abban különbözött a mai, modern leírástól, hogy Newton csak polinomok esetében használta ezt a módszert. Ő nem számolta ki az x_n rákövetkező közelítést, hanem kiszámolt egy sorozat polinomot és majd csak a végén ért el az x gyök közelítéséhez. Valószínűsíthető, hogy Newton a módszert Viéte egyik hasonló módszeréből vezette le. 1690-ben Joseph Raphson kiadott egy leegyszerűsített leírást. Raphson is polinomokkal dolgozott, azonban ő a módszert egymás után következő közelítések formájában írta le. Majd 1740-ben Thomas Simpson, a Newton-módszert iteratív módszerként tekinti, amely általános nemlineáris egyenletek megoldására szolgál.

Egy kis kitérő: a Newton-féle gyökvonó algoritmus

Elgondolkodtató, hogy régen milyen jól elboldogultak a számológép és számítógép nélküli időkben, mégis kitűnően tudtak számolni. Már a babiloniak is tudtak négyzetgyököt vonni. Az első közismert gyök a $\sqrt{2} = 1{,}41421\ldots$ az úgynevezett Pitagorasz állandó volt. A gyökvonás elvégzésére a numerikus matematikában ma is használt iterációs eljárást alkalmazták, amely sok számítógép és számológép működési elvének az alapja. Így gondolkodhatunk: ha egy *a* számból szeretnénk négyzetgyököt vonni, és tudjuk, hogy *b* közel van $\sqrt{a}$-hoz, akkor feltételezhetjük, hogy ha b valamennyivel kisebb $\sqrt{a}$-nál, akkor $\frac{b}{a}$ valamivel nagyobb lesz, így a számtani közepük talán *b*-nél jobban közelíti $\sqrt{a}$-t, vagyis lényegében a Newtonról elnevezett módszert használták.

Egy a pozitív szám négyzetgyökét a következő algoritmussal számolhatjuk ki:

Definiáljuk az x_n sorozatot az $x_0 = a$, $x_n = \frac{1}{2} \cdot \left(x_{n-1} + \frac{a}{x_{n-1}}\right)$, ha $n > 1$ rekurzióval.

Tétel: A fenti rekurzióval definiált x_n sorozat $\sqrt{a}$-hoz tart.

Írjunk hozzá egy olyan keretprogramot, ami tetszőleges valós szám esetén kiírja az iteráció egyes lépéseit, a hozzá tartozó értéket és a hibát is!

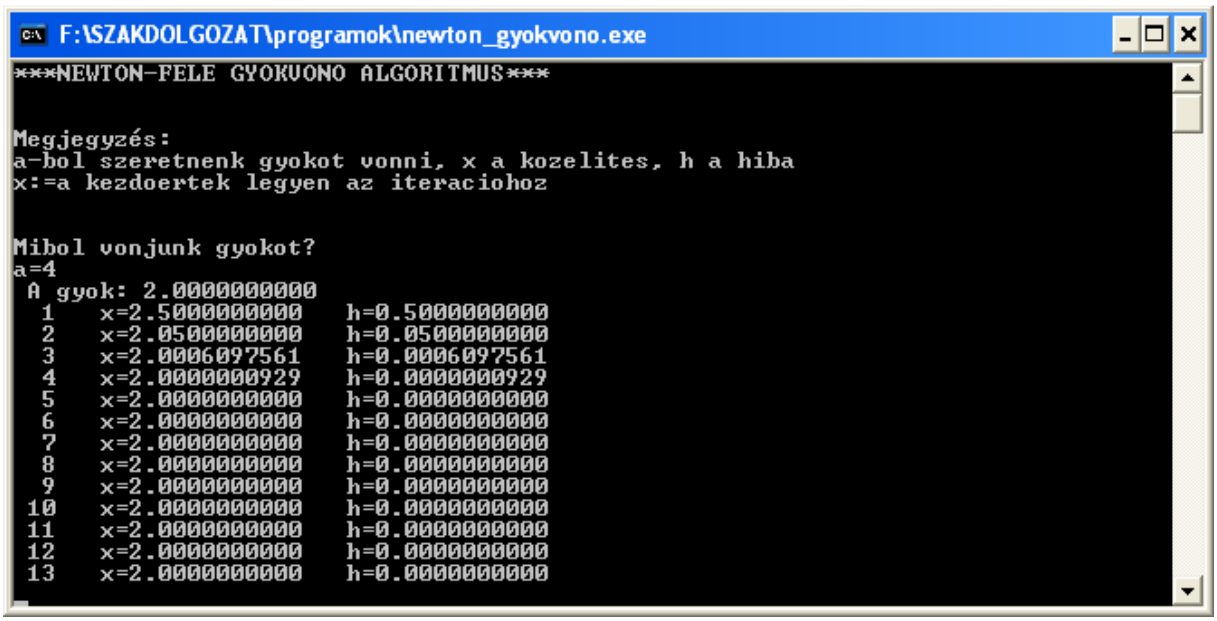
A változók deklarálása után, a programunk bekéri a-t, amiből négyzetgyököt szeretnénk majd vonni. Továbbá legyen x a közelítés, h pedig a hiba. A hibához persze kellene tudnunk $\sqrt{a}$ értékét, ezt b-ben tároljuk. A gyököt 10 tizedesjegy pontossággal írjuk ki a képernyőre. A cilusváltozó típusa egész legyen, míg a többi változóét deklaráljuk valós számnak. A program célja, hogy kiszámítsa b értékét a $\sqrt{x}$ utasítás használata nélkül. Habár a programban szerepel a `b:=sqrt(a)` sor, tehát a négyzetgyökvonás függvénye, azonban erre csak azért van szükség, hogy az eljárás hibáját nyomon tudjuk követni.

```
program gyokvonas;
uses crt;
var x,a,b,h:real;
    i:integer;
begin
  clrscr;
  writeln('***NEWTON-FELE GYOKVONO ALGORITMUS***');
  writeln;
  writeln;
  writeln('Megjegyzes:');
  writeln('a-bol szeretnenk gyokot vonni, x a kozelites, h a hiba');
  writeln('x:=a kezdoertek legyen az iteraciohoz');
  writeln;
  writeln;
  writeln('Mibol vonjunk gyokot?');
  write('a=');
  readln(a);
  x:=a;
  b:=sqrt(a);
  writeln(' A gyok: ',b:3:10);
  for i:=1 to 13 do
  begin
    x:=(x+(a/x))/2;
    h:=x-b;
    writeln(i:3,'  ',' x=',x:3:10,'   ','h=',h:3:10);
  end;
  readln;

end.
```

Teszteljük a programot!

Az $a = 4$ és $x_0 = a$ kezdőértékből induljunk ki. A futtatás eredményéből látható, hogy első közelítésként még $x \approx 2{,}5$-öt ír ki, majd az ötödik lépéstől olyan nagyon megközelíti a valós gyököt, vagyis a 2-t, hogy a hiba is elhanyagolható lesz.



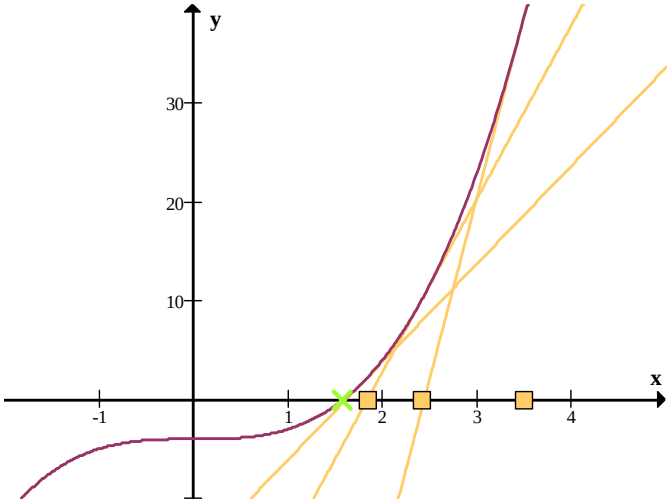
A Newton-módszer leírása [9][5][10][11][13][18]

Ötlet: induljunk ki egy, a gyökhöz közel található pontból, jelöljük α-val. A kezdőpont függvényértéke az $(\alpha, f(\alpha))$ pontban húzott érintőn található. Ezek után számoljuk ki az érintőnek az x tengellyel vett metszéspontját. Ez a pont egy jobb közelítése lesz a függvény gyökének, mint a kiinduló pontunk. A módszer fokozatosan közelítő lépések sorozatán keresztül közelíti meg a függvény gyökét.

Legyen $f: [a, b] \to \mathbb{R}$ függvény, keressük azt az $x^* \in [a, b]$ értéket, melyre $f(x^*) = 0$. Ezt az x^* értéket a függvény zérushelyének vagy gyökének nevezzük.

Algoritmus: Az első közelítő értéket jelöljük x_0-lal, amit grafikus módszerrel, vagy teljesen véletlenszerűen is kiszúrhatunk. Ezután az $y = f(x)$ görbét az $(x_0, f(x_0))$ pontjához húzott érintőjével közelítsük. Jelöljük x_1-gyel azt a pontot, amelyben ez az érintő elmetszi az x tengelyt. Kedvező esetben ez az x_1 szám már jobban közelíti a megoldást, mint az x_0. A

következő elem x_2 lesz, amely a görbe $(x_1, f(x_1))$ pontjához húzott érintő metszéspontja lesz az x tengellyel. Addig szükséges folytatnunk az algoritmust, amíg elég közel nem kerülünk a gyökhöz.



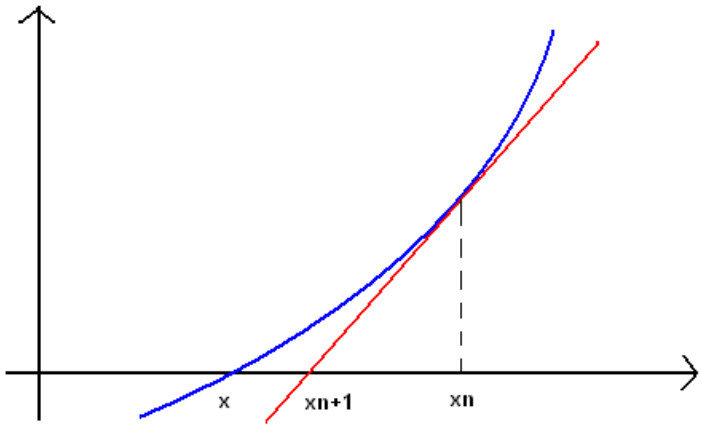
Legyen $\boldsymbol{f}: [\boldsymbol{a}, \boldsymbol{b}] \to \mathbb{R}$, $\boldsymbol{x_0} \in \mathbb{R}$ adott kezdőérték. Feltételezhető, hogy $\boldsymbol{f}$ folytonosan differenciálható függvény.

A közelítésre megadhatunk egy képletet is. A közelítést jelöljük x_n-nel. Az x_n közelítés ismeretében az $y = f(x)$ görbét közelítsük az $(x_n, f(x_n))$ pontokon átmenő $f'(x_n)$ meredekségű érintő egyenessel[1]. Ha adott az x_n közelítés, akkor a görbe $(x_n, f(x_n))$ pontjához húzott érintő: $y = f(x_n) + f'(x_n)(x - x_n)$. Ennek x tengellyel való metszéspontjának meghatározásához y-t nullával tesszük egyenlővé.

$$\left.\begin{matrix} y = f(x_n) + f'(x_n)(x - x_n) \\ y = 0 \end{matrix}\right\}$$

egyenletrendszert kell megoldani: $\boldsymbol{x = x_n - \frac{f(x_n)}{f'(x_n)}}$, ahol $\boldsymbol{f'(x_n) \neq 0}$ és az $\boldsymbol{x}$ érték lesz a következő közelítés. Tehát az egyenes $\boldsymbol{x}$ tengellyel vett metszéspontját jelöljük $\boldsymbol{x_{n+1}}$-gyel.

[1] A koordináta-rendszer $(a, f(a))$ és a tőle különböző $(x, f(x))$ pontjain át fektessünk egy szelőt. Az egyenes meredeksége, más néven iránytangense $\frac{f(x)-f(a)}{x-a}$. Ha $x \to a$-hoz, akkor a szelők tartanak egy határhelyzethez, amit érintőnek nevezünk, így a szelők meredeksége is tart az érintő meredekségéhez. Ezt a határértéket nevezzük deriváltnak.



A Newton-módszer geometriai jelentése

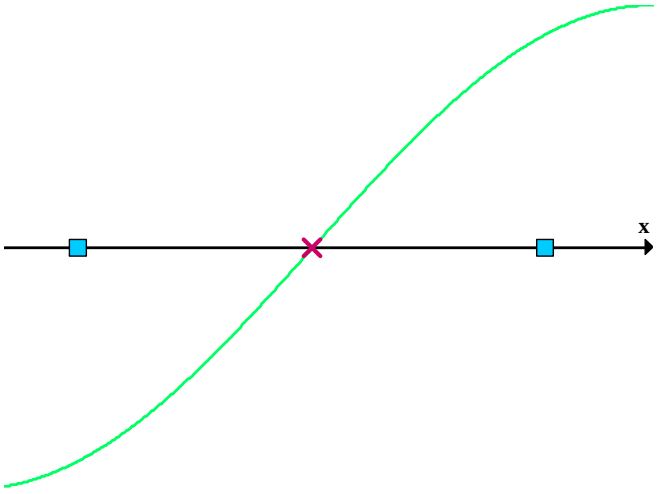
Általában az x_0 kezdőérték adott. Ekkor a formula: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, ahol $n \in \mathbb{N}$, ha $f'(x_n) \neq 0$.

A Newton-módszer konvergenciája

A gyakorlatban nagyon eredményesen, nagyon gyorsan konvergál a módszer. Azonban erre nincs garancia. A konvergenciát biztosítja a következő eredmény:

Tétel:[12] ha egy x^* gyököt tartalmazó intervallum minden x elemére teljesül, hogy $\left|\frac{f(x)\cdot f''(x)}{(f'(x))^2}\right| < 1$, akkor a Newton-módszer konvergál az x^* gyökhöz, amennyiben a kezdőérték beleesik ebbe az intervallumba.

Megjegyzésként megemlíteném, a Newton-módszer mindig konvergál, ha x^* és x_0 között az f grafikonja konvex, ha $f(x_0) > 0$, illetve konkáv, ha $f(x_0) < 0$.



Tétel: [14] Azt mondjuk, hogy az (x_n) sorozat x_0-ból indítva, legalább K-ad rendben konvergál, ha $K \geq 1$, és van olyan nem negatív C konstans, hogy érvényes az

$$|x_{n+1} - x^*| \leq C \cdot |x_n - x^*|^K, \qquad n = 0,1, \dots$$

egyenlőtlenség.

A tételt a Newton-módszerre általánosítva:

$$|x_{n+1} - x^*| \leq \frac{max|f''|}{2min|f'|} |x_n - x^*|^2 = const \cdot |x_n - x^*|^2$$

ahol $min|f'|$ és $max|f''|$ az x^* szám valamely környezetében felvett maximális, illetve minimális érték. Szavakkal a képlet azt fejezi ki, hogy az $n+1$-edik lépésben a hiba nem lehet nagyobb, mint az n-edik lépés hibanégyzetének konstansszorosa. Tehát a Newton-módszer kvadratikusan (másodrendben) lesz konvergens, melyhez szükségesek azok a feltételek, hogy x_0 elég közel legyen a gyökhöz és $f'(x_0) \neq 0$. A sorozat konvergencia rendjéről szóló tétel szerint tehát a felezési módszer lineárisan (első rendben) lesz konvergens, tehát lassabban közelíthetjük meg a keresett gyököt, mind a Newton-módszerrel. Gyakorlatban a konvergencia lokális, és nem ismerjük előre a konvergencia környezetét.

Definíció: Egy iterációs eljárásról akkor mondjuk, hogy lokálisan konvergál az x^*-hoz, ha van olyan pozitív δ sugarú $\mathcal{g}(x^*, \delta)$ gömb, amelyik a következő tulajdonságokkal rendelkezik:

- tetszőleges $x_0 \in \mathcal{g}(x^*, \delta)$ esetén az iteráció által előállított (x_n) sorozat konvergál x^*-hoz
- $\mathcal{g}(x^*, \delta)$-n kívül viszont van olyan x_0 vektor, amelyre ez nem igaz.

Ekkor $\mathcal{g}(x^*, \delta)$ az x^* vonzási gömbje.

A egyszerűség kedvéért jelöljük $m_1 = min|f'|$-tal és $M_2 = max|f''|$-tal.

Tétel *(Lokális konvergencia tétel)***:**

Ha $f \in \mathbb{C}^2[a, b]$,

1. $\exists x^* \in (a, b): f(x^*) = 0$,
2. f' állandó előjelű az $[a, b]$-n,
3. $0 < m_1 \leq |f'|$,
4. $|f''| \leq M_2 < \infty$,
5. legyen $x_0 \in [a, b]$ olyan, hogy $|x_0 - x^*| < r \coloneqq min\left\{\frac{2 \cdot m_1}{M_2}, |x^* - a|, |x^* - b|\right\}$.

Ekkor az x_0-ból indított Newton-módszer másodrendben konvergál az f gyökéhez. Mivel a konvergencia kvadratikus, így a hiba négyzetesen csökken, így a helyes jegyek száma megduplázódik minden lépésnél.

Azonban vannak eredmények globális konvergencia esetén is.

Tétel *(Globális vagy monoton konvergencia tétel)***:**

Ha $f \in C^2[a,b]$,

1. $\exists x^* \in (a,b): f(x^*) = 0$,
2. f' és f'' állandó előjelű az $[a,b]$-n,
3. $x_0 \in (a,b): f(x_0) \cdot f''(x_0) > 0$,

akkor az x_0-ból indított Newton-módszer (x_n) sorozata monoton konvergál f gyökéhez.

Többszörös gyökök közelében az algoritmus konvergencia rendje csak lineáris. Erre a speciális esetre létezik a módszernek módosítása, amit tompított Newton-módszernek nevezünk. A Newton-módszer esetében újabb közelítés számításakor f és f' egy-egy függvényértékét kell kiszámítanunk. Ha f' a gyök környezetében alig változik, akkor a Newton-formulába $f'(x_n)$ helyett $f'(x_0)$ értékét írhatjuk. Ezzel a módszer műveletigényét jelentősen lecsökkentjük, mivel f' értékét, csak az iteráció kezdésekor kell kiszámolnunk. Tehát a tompított (vagy módosított) Newton-módszer formulája: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_0)}$.

A Newton-módszer: pro és kontra [5][9][11]

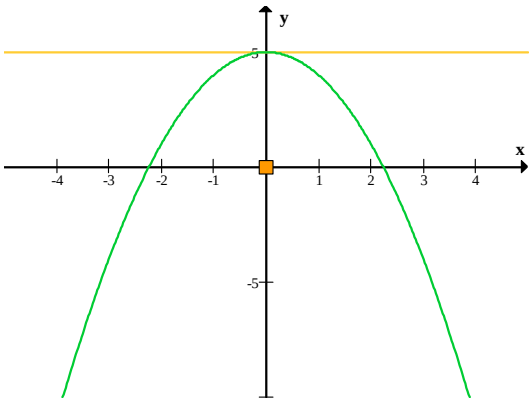
Matematika egyik fő problémája az egyenletek megoldása. Ismerjük a másodfokú egyenletek megoldására a megoldóképletet illetve egy-két bonyolultabb módszert a harmad- illetve negyedfokú egyenletek gyökeinek megkeresésére is (például Cardano képlet). Niels Henrik Abel megmutatta, hogy nem létezik egyszerű megoldóképlet az ötödfokú és az olyan egyenletekre sem, amelyekben transzcendens függvények mellett polinomok vagy más algebrai függvények szerepelnek.

Definíció: transzcendens függvény: minden nem algebrai függvényt így nevezünk. Eszerint a trigonometrikus függvények, azok inverzei, az exponenciális és logaritmikusfüggvények, valamint a hiperbolikus függvényeket is idesoroljuk.

Később Évariste Galois megmutatta, hogy az ötnél magasabb fokú esetekben sem létezik megoldóképlet. Persze ez nem azt jelenti, hogy nincs megoldás, hanem, hogy nincs olyan véges sok lépés után véget érő eljárás, amely csak a négy algebrai műveletet illetve a gyökvonást használja a gyökök megkeresésére. A Newton-módszer segítségével közelítő megoldást kaphatunk az $f(x) = 0$ egyenlet megoldására, mint az egyváltozós egyenletekre $(f: \mathbb{R} \to \mathbb{R})$, és a nemlineáris egyenletrendszerekre egyaránt $(f: \mathbb{R}^n \to \mathbb{R}^n, n \geq 2, x \in \mathbb{R}^n, \text{f folyt.diff.})$.

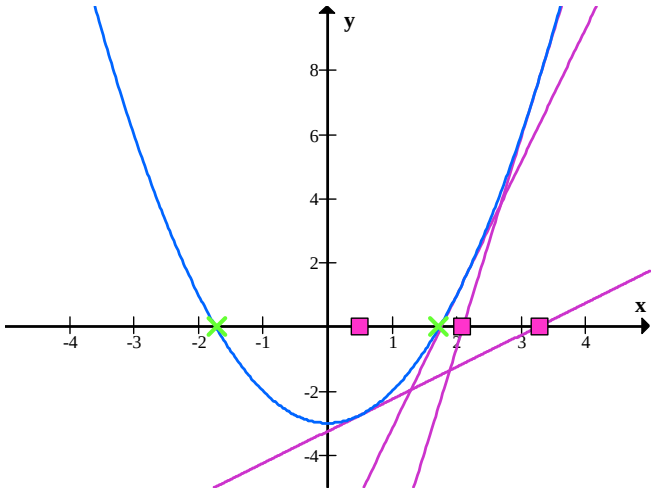
Nézzük az algoritmus hátrányait!

Ha az $f'(x_n) = 0$-val, akkor ezzel a módszerrel nem tudunk továbblépni, hiszen nem lesz metszéspont az x tengellyel, amely x_{n+1}-et kijelölné. Ilyenkor új kezdőértékből kell indítanunk az iterációt.



A Newton-módszerhez szükséges a derivált kiszámolása. Létezik egy hatékonyabb, úgynevezett húrmódszer[1] is, ahol a deriváltat csak megközelítjük a függvény két pontján áthaladó ferde egyenessel. Rendszerfejlesztők a húrmódszert alkalmazzák, nemcsak a számításokhoz szükséges kevesebb erőfeszítések miatt, hanem mert a fő szempont a hatékonyság, így a kevesebb kód elsődlegesebb, mint egy másodrendű konvergencia.

Előfordulhat olyan is, hogy az (x_n) sorozat konvergál egy gyökhöz, azonban előfordulhat, hogy nem a kívánt gyökhöz.



Ha x^* többszörös gyök, akkor $f'(x^*) = 0$ vagyis, ha $x_n \approx x^*$, akkor $f'(x_n) \approx 0$. Ilyenkor a Newton-módszer lineárisan konvergens, tehát lassú. A következőkben nézzünk példát

[1] Húrmódszer
x_0 ,x_1 adott, $f(x_0) \cdot f(x_1) < 0$. Ekkor $x_{n+1} := x_n - \frac{f(x_n)\cdot(x_n - x_k)}{f(x_n) - f(x_k)}$, ahol k a legnagyobb index, melyre $f(x_n) \cdot f(x_k) < 0$.

azokra az esetekre, ahol a Newton-módszer használata közben az eredmények megtévesztőek lehetnek.

1. Példa: Távoli kezdőpont

Ha a kezdőpont nincs elég közel a gyökhöz, a konvergencia elmaradhat.

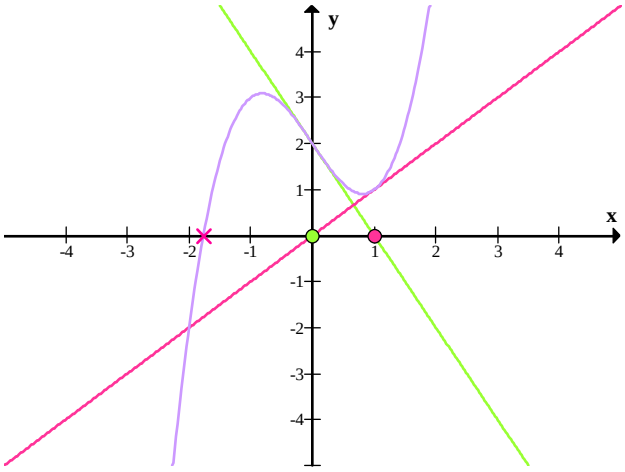
Nézzük a következő függvényt: $f(x) = x^3 - 2x + 2$. $f'(x) = 3 \cdot x^2 - 2$. A kezdeti pont legyen: $x_0 = 0$. Az iteráció:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0 - \frac{0^3 - 2 \cdot 0 + 2}{3 \cdot 0^2 - 2} = 0 - \frac{0 - 0 + 2}{0 - 2} = 1$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 1 - \frac{1^3 - 2 \cdot 1 + 2}{3 \cdot 1^2 - 2} = 1 - \frac{1 - 2 + 2}{1 - 2} = 0$$

$$x_3 = x_2 - \frac{f(x_2)}{f'(x2)} = 0 - \frac{0^3 - 2 \cdot 0 + 2}{3 \cdot 0^2 - 2} 0 - \frac{0 - 0 + 2}{0 - 2} = 1$$

Tehát a folyamat oszcillálni fog a két érték között. Nem éri el a gyököt.

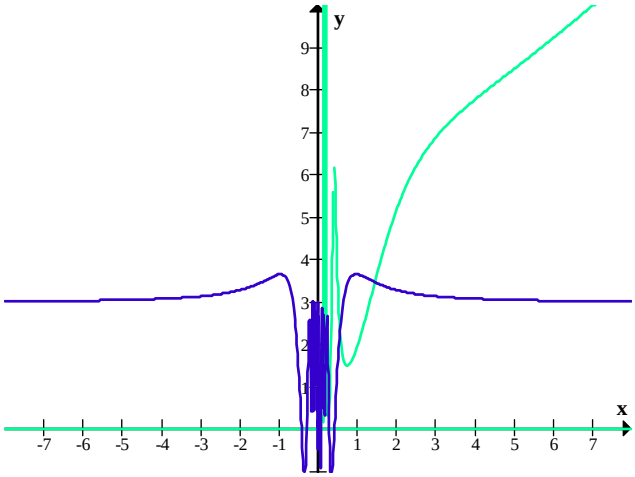


$f(x) = x^3 - 2x + 2$

2. Példa: Ha a derivált nem folytonos

Ha a derivált nem folytonos a gyöknél, akkor a módszer nem fog konvergálni, akármilyen intervallumot is keresünk a gyök számára. Például: $f(x) = \begin{cases} 0 \; ha \; x = 0 \\ x + x^2 sin\left(\frac{2}{x}\right) ha \; x \neq 0 \end{cases}$

Deriváltja a 0-ban: $f'(x) = 1 + 2x \cdot sin\left(\frac{2}{x}\right) - 2 \cdot cos\left(\frac{2}{x}\right)$ és $f'(0) = 1$. Bármely intervallumot is veszünk ez a derivált változtatni fogja az előjelét, mihelyt *x* megközelíti a 0-t jobbról, illetve balról. Tehát $\frac{f(x)}{f'(x)}$ végtelen a gyök közelében, így a Newton-módszer nem fog konvergálni.



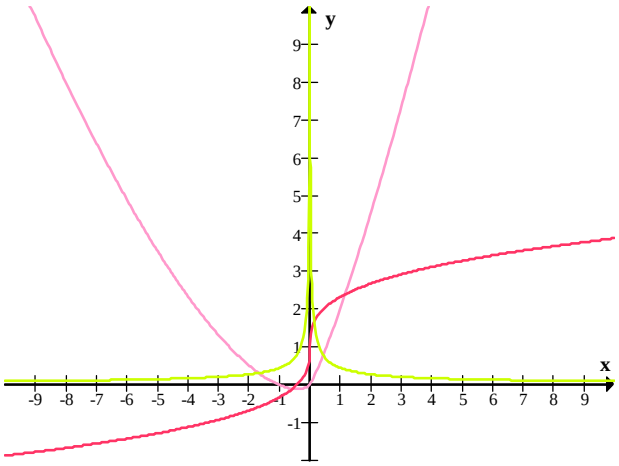
$$f(x) = \begin{cases} 0 & \text{ha } x = 0 \\ x + x^2 \cdot \sin\left(\frac{2}{x}\right) & \text{ha } x \neq 0 \end{cases} \text{ és deriváltja } f'(x) = 1 + 2x \cdot \sin\left(\frac{2}{x}\right) - 2\cos\left(\frac{2}{x}\right)$$

3. Példa: Második derivált hiánya

Ha nem létezik a gyöknél a második derivált, akkor előfordulhat, hogy a konvergencia nem lesz kvadratikus. Tekintsük az $f(x) = x + x^{\frac{4}{3}}$ függvényt és első deriváltját:

$f'(x) = 1 + \frac{4}{3} \cdot x^{\frac{1}{3}}$, majd a másodikat: $f''(x) = \frac{4}{9} \cdot x^{\frac{-2}{3}}$ (ha $x = 0$, akkor nem létezik).

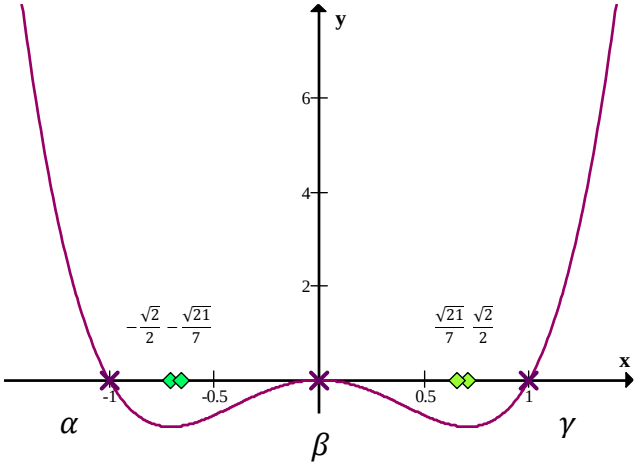
Tegyük fel, hogy ismerjük x_n-et. Ekkor $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = \frac{\frac{1}{3} \cdot x_n^{\frac{4}{3}}}{1 + \frac{4}{3} \cdot x_n^{\frac{1}{3}}} \approx \frac{4}{3}$. Ebben az esetben a Newton-módszer konvergenciája nem lesz kvadratikus. Ha a derivált nulla a gyökben, akkor a konvergencia szintén nem lesz kvadratikus. Például: ha $f(x) = x^2$, akkor $f'(x) = 2x$. Képlet szerint $x - \frac{f(x)}{f'(x)} = x - \frac{x}{2} = \frac{x}{2}$.



$$f(x) = x + x^{\frac{4}{3}},\ f'(x) = 1 + \frac{4}{3}x^{\frac{1}{3}},\ f''(x) = \frac{4}{9}x^{\frac{-2}{3}}$$

4. Példa: Fraktálmedencék

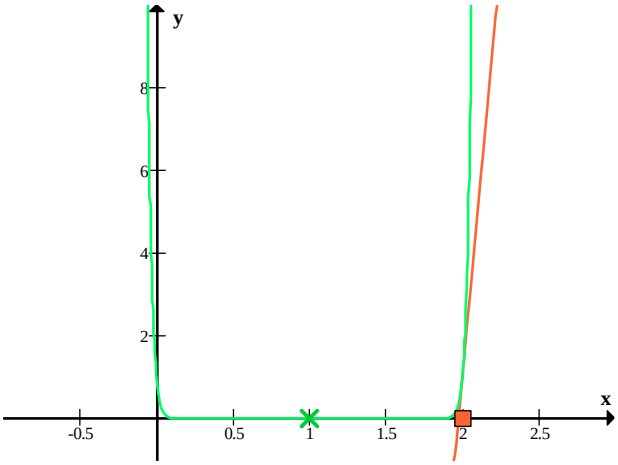
Nagyon oda kell figyelni a kezdeti érték megválasztására. Nézzük a $4x^4 - 4x^2 = 0$ egyenletet. Ha az x tengely $\left(-\infty; -\frac{\sqrt{2}}{2}\right)$ intervallumából választunk kezdőértéket, akkor az α gyökhöz jutunk. Ha a $\left(-\frac{\sqrt{21}}{7}, \frac{\sqrt{21}}{7}\right)$-ből akkor β-t kapjuk meg, végül pedig, ha $\left(\frac{\sqrt{2}}{2}, \infty\right)$ akkor γ-hoz konvergál a sorozat. A $\frac{\sqrt{21}}{7}$ és $\frac{\sqrt{2}}{2}$ intervallumban végtelen sok olyan nyílt intervallum található, amelynek pontjai mind az α gyökhöz vezetnek, azonban ezek között végtelen sok olyan nyílt intervallum helyezkedik el, amelyeknek a pontjai a γ gyökhöz tartanak. Hasonló a helyzet a $\left(-\frac{\sqrt{2}}{2}, -\frac{\sqrt{21}}{7}\right)$ intervallumban is. Ezek szerint felfoghatjuk a gyököket úgy, mint egyfajta vonzási centrumokat is, melyek vonzzák a pontokat. Azt gondolnánk, hogy α és β gyökök közé eső pontok vagy az egyik gyökhöz, vagy a másikhoz vonzódnának azonban a két pont között található végtelen olyan sok intervallum, amelynek pontjai γ-hoz közelednek. A medencék határán bonyolult alakzatok ismétlődnek vég nélkül. Az ilyen medencéket nevezzük fraktálmedencéknek.



Megjegyzés: Ha az $f(x) = 4x^4 - 4x^2$ függvény mindhárom gyökét meghatározhatjuk, ha a kezdeti értéket $x = \frac{\sqrt{21}}{7}$ közelében vesszük fel.

5. Példa: A gyök környezetében sima görbék

Léteznek olyan esetek is, amikor a gyök környezetében a függvénygörbék annyira közelítenek a vízszinteshez, hogy a Newton-módszer használható becslés nélkül ér véget. Nézzük meg a $f(x) = (x-1)^{40}$, $x_0 = 2$ kezdeti értékkel! (A gyök $x = 1$-ben lesz.)

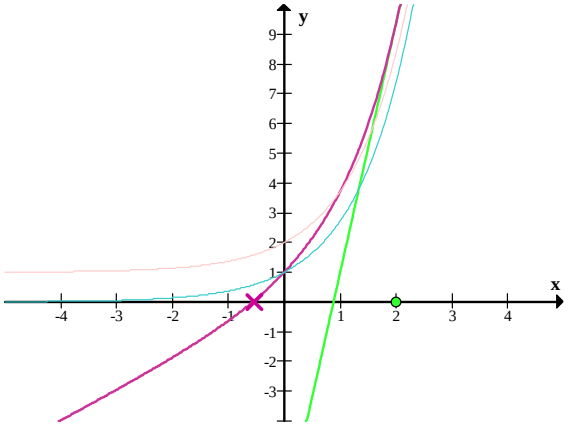


Feladatok: Newton-módszer alkalmazásával

1. Példa [9]

Az $f(x) = e^x + x = 0, (x \in \mathbb{R})$ megoldására írjuk fel a Newton módszert. Vizsgáljuk meg a két konvergencia tétel segítségével, hogy milyen x_0 -ra konvergens!

$x_{n+1} \coloneqq x_n - \frac{e^{x_n}+x_n}{e^{x_n}+1} = x_n - 1 - \frac{x_n-1}{e^{x_n}+1} = (x_n - 1)\left(1 - \frac{1}{e^{x_n}+1}\right) = (x_n - 1)\frac{e^{x_n}}{e^{x_n}+1}.$



$\boldsymbol{f(x) = e^x + x, f'(x) = e^x + 1, f''(x) = e^x}$

- Először vizsgáljuk a globális konvergencia tétel feltételeit!
    1. Például $[a, b] = [-10, +10]$-ben van gyöke az egyenletnek, mert: $f(-10) \cdot f(10) < 0$ (Bolzano tétel)
    2. $f'(x) = e^x + 1 > 0$ és $f''(x) = e^x > 0$ $(\forall x \in \mathbb{R})$
    3. Mivel $f'' > 0$, ha $x_0 \in [-10, +10]$-t úgy választjuk, hogy $f(x_0) > 0$, akkor a Newton módszer (x_n) sorozata monoton fogyóan konvergál f gyökéhez. $(x_0 > x^*)$
- Nézzük meg a lokális konvergencia tétellel is, hogy lássuk más feltételek mellett milyen eredményt kapunk. Célunk, hogy minél nagyobb környezetet meg tudjuk adni, ezért célszerű m_1-et a lehető legnagyobbnak, M_2–t a legkisebbnek választani.

1. Például $[a,b] = [-10,0]$ esetén $f(-10) \cdot f(0) < 0$. Ebben az intervallumban van gyöke az egyenletnek.
2. $f'(x) = e^x + 1 > 1 \ \ (x \in \mathbb{R})$
3. $|f'(x)| \geq |e^x + 1| \geq 1 = 1$ így $m_1 = 1$
4. $|f''(x)| \leq |e^x| \leq 1$, ha $x < 0$, így $M_2 = 1$
5. Tehát, ha $x_0 \in [-10,0]$ olyan, hogy

$$|x_0 - x^*| < r := min\left\{\frac{2 \cdot 1}{1}, |x^* + 10|, |x^* - 0|\right\}$$

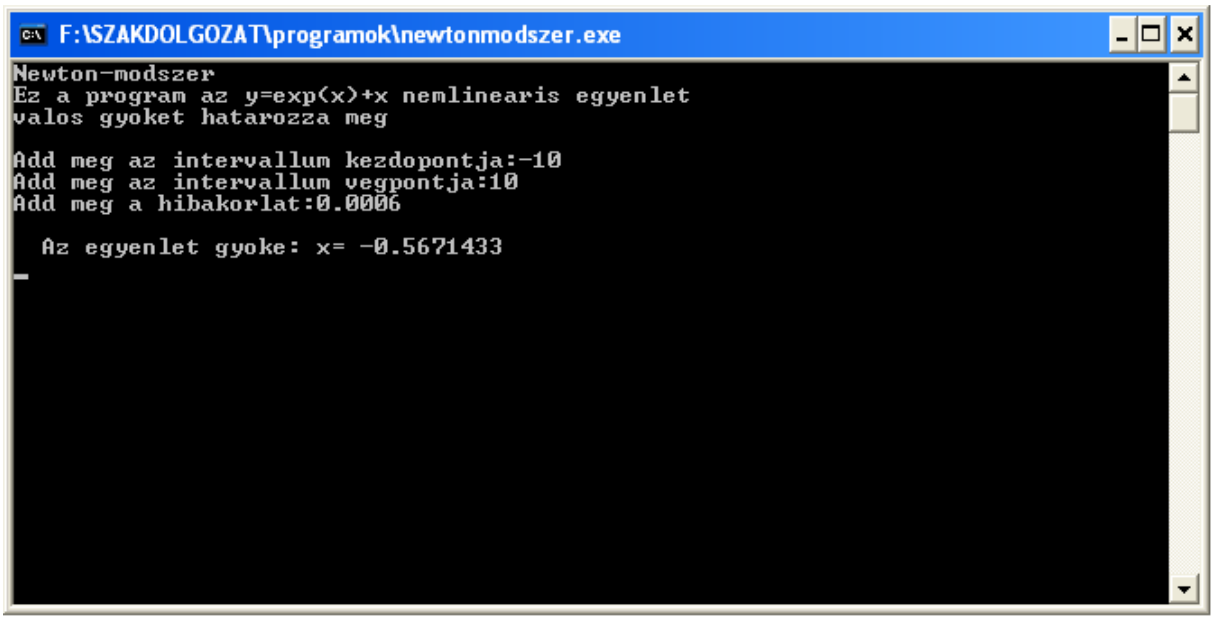
akkor az x_0-ból indított Newton-módszer másodrendben konvergál az f gyökéhez.

Program:

A program segítségével ellenőrizhetjük, hogy valóban jól gondolkodtunk-e. Adott függvényre a program visszaadja a gyököt a Newton-módszer felhasználásával. A program lényegi része a következő:

```
function newtonm(x,eps:real; var r:boolean; f,fd:fv):real;
var kertek,nev:real;
begin
 h:=true;
 repeat
  kertek:=x;
  nev:=fd(kertek);
  if abs(nev)>1.e-5 then
    x:=kertek-f(kertek)/nev
      else
        h:=false;
 until (abs(x-kertek)<hiba) or (not r);
 if h then newtonm:=x;
end;
```

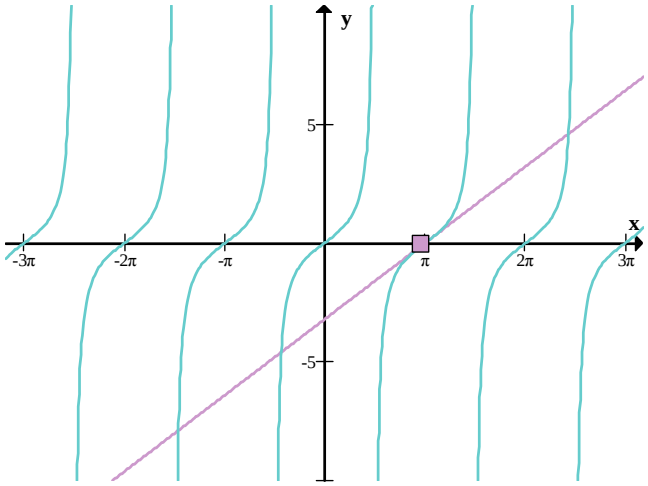
A függvény magyarázata röviden: a kezdőértéket egyenlővé tesszük x-el, majd ezt a kezdőértéket behelyettesítjük a derivált függvényünkbe. Így megkapjuk a nevezőt. Ebből már fel tudjuk írni a Newton-módszer formuláját: `x:=kertek-f(kertek)/nev` mindezt egy hátultesztelős ciklusban megfogalmazva, ami addig tart, amíg hibát nem kapunk: `not r`, illetve amíg `abs(x-kertek)<hiba.` Ez azt jelenti, hogy ha az x és a kezdőérték különbsége kisebb, mint a megadott hibakorlát, akkor leáll a program. Végül, ha nincs hiba, akkor a megoldást, vagyis az egyenlet valós gyökét az x változó adja meg. A teljes program megtalálható a mellékelt CD-n newtonmodszer.pas néven.



Ha megnézzük a grafikont, láthatjuk, hogy valóban, valahol -0.5 környékén található az egyenlet gyöke.

1. Példa[11]

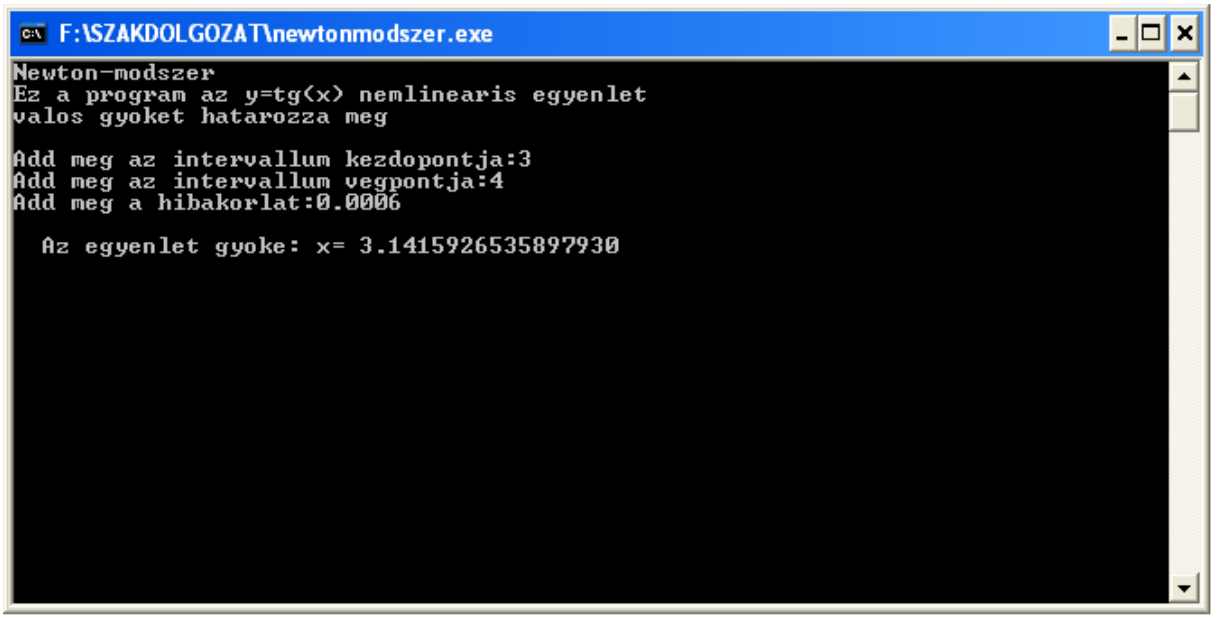
Határozzuk meg π közelítő értékét a Newton-módszerrel annyi tizedesjegyre, amennyit kalkulátorunk kijelzője meg tud jeleníteni oly módon, hogy megoldjuk $x_0 = 3$ kezdő értékkel a $tg(x) = 0$ egyenletet.



Az ábrán a tg függvényt ábrázoltuk, illetve egy az $x_0 = 3$ kezdőpontból húzott érintőt. Ezzel könnyen meg tudjuk becsülni π értékét, hiszen minden iterációs lépésnél egyre jobban közelítjük meg π-t.

Az előző program segítségével kiszámolhatjuk ezt is. Mivel a PASCAL nem ismeri a tangens függvényt, ezért fel kell használnunk a programban a $tg(x) = \frac{\sin(x)}{\cos(x)}$ azonosságot, mikor

függvényként megadjuk. A deriválttal már nem lesz probléma, hiszen abban nem szerepel már a tangens, mint függvény. ($tg(x)' = \frac{1}{\cos^2(x)}$)



Hatékonyság vizsgálata: egy másik módszer: az intervallumfelező eljárás

[9][12][13]

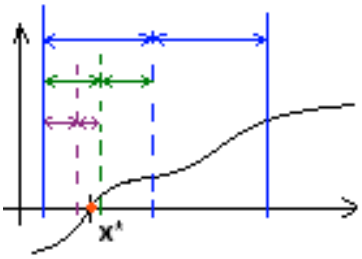
Legyen adott egy $f: \mathbb{R} \to \mathbb{R}$ függvény.

Hatékonyan megközelíthetjük az $f(x) = 0$ egyenlet (egyik) gyökét a felezési algoritmussal.

Tétel *(Bolzano tétele)***:** Ha $f \in C[a,b]$ és $f(a) \cdot f(b) < 0$, akkor *f*-nek van gyöke $[a,b]$-ben.

Kezdőértékek legyenek az $x_0 := a$ és $y_0 := b$. Definiáljuk x_{n+1}, y_{n+1}-et a következő rekurzióval:

- Ha $f(x_n) \cdot f\left(\frac{x_n+y_n}{2}\right) < 0$, akkor $y_{n+1} := \frac{x_n+y_n}{2}$.
- Ha $f(x_n) \cdot f\left(\frac{x_n+y_n}{2}\right) = 0$, akkor $x^* := \frac{x_n+y_n}{2}$ a keresett gyök.
- Ha $f(x_n) \cdot f\left(\frac{x_n+y_n}{2}\right) > 0$, akkor $x_{n+1} := \frac{x_n+y_n}{2}$.

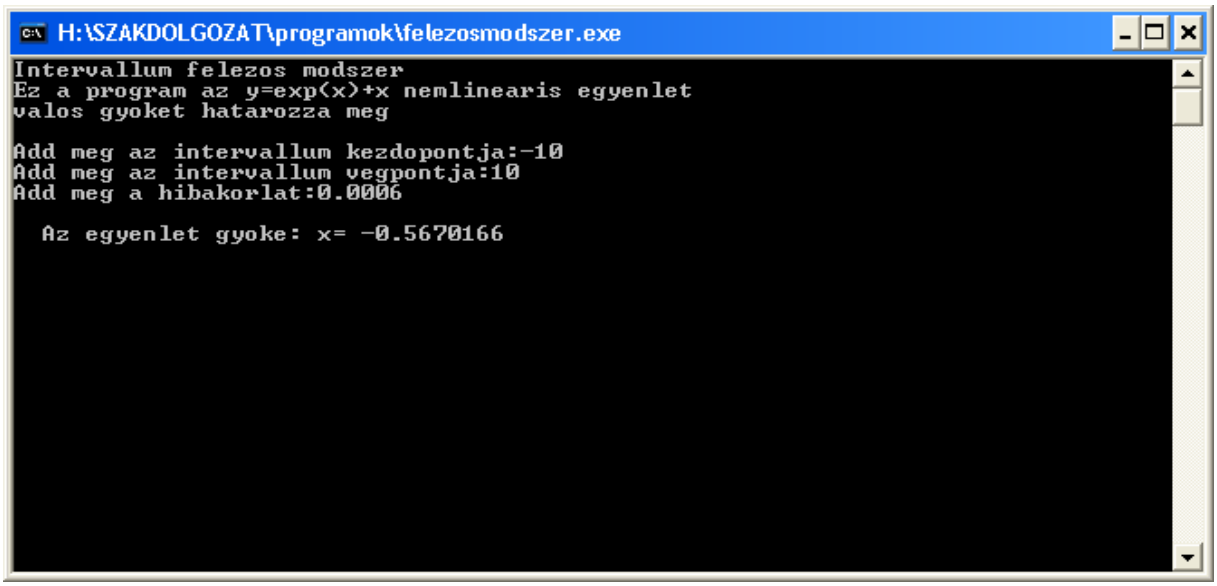


Az egymásba skatulyázott zárt intervallumok ráhúzódnak az egyenlet gyökére. Ez egy másik gyökkeresési módszer. Vajon melyik a jobb? Első ránézésre az intervallumfelező eljárás egyszerűbbnek tűnhet. Ha nincs szükségünk nagy pontosságra, akkor még gyorsnak is mondható. Probléma lehet, hogy a gyök közelében a kerekítési hibák miatt az $f(x)$ előjele hibás lehet. Gyenge pont lehet még a kiindulási helyzet megkeresése. Ezt megkönnyíthetjük: amelyik intervallumban várjuk a

gyököt, abban több pontra is kiszámítjuk a függvényt, így ez által megkereshetjük az előjelváltási helyeket. Egyébként minden lépésben a bizonytalansági intervallumot[1] a felére csökkentjük. Előnye, hogy nem szükséges a függvények deriváltjainak az ismerete, vagy a deriváltak létezése. Hátránya a Newton-módszerrel szemben, hogy több lépésben éri el a megkövetelt pontosságot, hiszen konvergenciarendje csak lineáris (elsőrendű). Mivel a módszer konvergenciája lassú, ezért a gyök közelítő értékének behatárolására használjuk.

Program:

```
function intfelezo(a,b,hiba:real; var h:boolean; f:fv):real;
var fa,fb,x,fx:real;
begin
 fa:=f(a);
 fb:=f(b);
 if (fa*fb < 0) then
   begin
    fx:= abs(fa);
    while fx >hiba do
     begin
      x:=(a+b)/2;
      fx:=f(x);
      if fx*fa > 0 then begin
                          a:=x;
                          fa:=fx;
                        end
                   else b:=x;
      fx:=abs(fx);
   end;
   h:=true;
   intfelezo:=x;
   end
   else h:=false;
end;
```



[1] Az $f(x) = 0$ egyenlet x_n ε közelítő megoldása azt jelenti, hogy x^* gyök az x_n ε sugarú környezetében van, vagyis $|x_n - x^*| < \varepsilon$ ahol ε egy előre rögzített érték, a megengedett hiba. Bizonytalansági intervallumnak $(x_n - \varepsilon, x_n + \varepsilon)$-ot nevezzük.

Rekurzió egy fizikai alkalmazása: Ballisztikus görbe

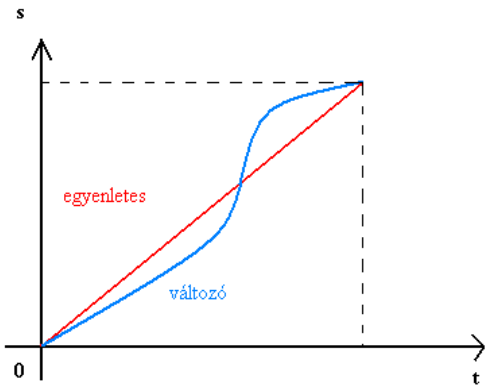
Differenciálegyenletek, differenciaegyenletek és a fizika [1][11][14][15][16][17]

A fizika egyik fontos területe a mozgások leírása. A testek mozgása lehet haladó mozgás, forgómozgás vagy a haladó és a forgómozgás kompozíciójából az összetett mozgás. A haladómozgások leírására be kell vezetnünk néhány mennyiséget:

- $t \mapsto s(t)$, vagyis a megtett út az idő függvényében és
- $t \mapsto v(t)$, a sebesség az idő függvényében, vagyis a t időpontbeli pillanatnyi sebesség,
- $t \mapsto a(t)$, a gyorsulás az idő függvényében, tehát a t időpontbeli pillanatnyi gyorsulás.

Ha a valóság egy időben lejátszódó folyamatát szeretnénk matematikailag megfogalmazni, akkor ha az időt rövid időközökre felbontjuk, gyakran az n-edik időközben lejátszódó változás meghatározható lesz az $n-1$-edikből. Valójában itt lelhetjük fel a rekurziót, mint fogalmat a fizikában, hiszen ezzel az elgondolással egy másodrendű rekurziót írhatunk fel.

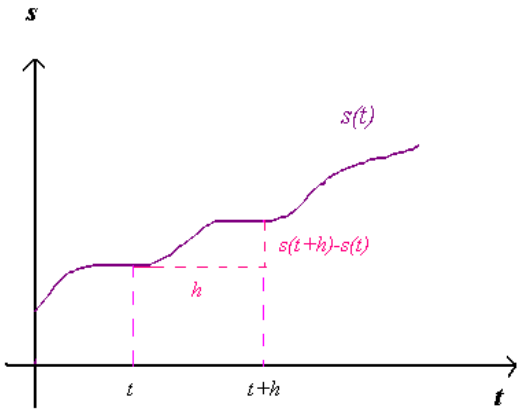
A differenciálás, a differenciálhányados fogalmának a bevezetését egy fizikai interpretáció segítségével is lehet szemléltetni. Az olyan mozgást, amely során a test egyenlő időtartamok alatt egyenlő utakat jár be egyenletes mozgásnak szokás nevezni. A mozgás során megtett út egyenesen arányos az út megtételéhez szükséges idővel, vagyis minden egyenletesen mozgó testnél a $\frac{\Delta s}{\Delta t}$ hányados állandó és annál nagyobb, minél gyorsabban mozog a test. Ez a hányados alkalmas mennyiség a testek egyenletes mozgásainak jellemzésére, neve: sebesség. A természetben és a gyakorlatban lezajló mozgások többsége nem egyenletes mozgás, hanem változó mozgás. Kezdőpontként rögzítsünk egy 0 pontot az időben és az egyenesen. Jelöljük $s(t)$-vel egy pontnak elfoglalt helyzetét a t időpillanatban 0-tól, tehát a pont mozgását a $t \mapsto s(t)$ út-idő függvénnyel írjuk le. Rögzítenünk kell egy rövid időközt, amit jelöljünk h-val!



A két test egyenlő idők alatt egyenlő utakat tett meg, mozgásuk mégis különböző volt

A mozgás t időpontban lévő sebességének a kiszámítása:

Ha a mozgó pont a t időpontig $s(t)$, akkor a $t+h$ időpontig nyilvánvalóan $s(t+h)$ utat tesz meg. Ekkor az előre rögzített h intervallumban $s(t+h)-s(t)$ lesz a megtett út. Mennyi a $[t, t+h]$ intervallumra eső átlagsebesség? A változó mozgást végző test esetében is az egyenletes mozgásra megismert sebesség kiszámítási módot alkalmazzuk akkor a megtett $s(t+h)-s(t)$ út és a megtételéhez szükséges $t+h-t=h$ idő hányadosa lesz. Átlagsebességen tehát azt a sebességet értjük, amellyel a test egyenletesen mozogva ugyanazt az utat ugyanannyi idő alatt tenné meg, mint változó mozgással. Képletben az átlagsebesség: $v(t)=\frac{s(t+h)-s(t)}{h}$, amit az s függvény $s(t)$ pontbeli differenciahányadosának nevezünk.[1] Ezt gyakran $\frac{\Delta s}{\Delta t}$ jelöli. Az átlagsebesség nem ad információt a test útközbeni helyzetéről. Ezért egy új mennyiség megalkotására van szükség, a pillanatnyi sebességre. A differenciahányados minden t rögzített időpontban egy rögzített értékhez tart, ha $h \to 0$-hoz, vagyis ha vesszük a differenciahányados határértékét, megkapjuk a függvény differenciálhányadosát, ami pedig megadja a pillanatnyi sebességet. Vagyis: $\lim_{h\to 0}\frac{s(t+h)-s(t)}{h}=v(t)$ adja a t időpontban lévő pillanatnyi sebességet. Fizikailag tehát a sebességet az út-idő függvény differenciálhányados függvényeként definiáljuk.



Hasonlóan definiáljuk a gyorsulást is:

A fizikában a gyorsulás, melyet a-val jelölünk, a sebesség változási gyorsasága. Egyenletesen változó mozgásnál egyenlő időtartam alatt mindig ugyanannyival változik a sebesség, vagyis ahányszor hosszabb a változás időtartama, annyiszor nagyobb a sebességváltozás. A sebességváltozás egyenletesen arányos a közben eltelt idővel. Ebből

[1] Az $\frac{f(b)-f(a)}{b-a}$ differenciahányados megegyezik az $(a, f(a))$ és $(b, f(b))$ pontokon átmenő egyenes meredekségével.

következik, hogy $\frac{\Delta v}{\Delta t} =$ állandó. Definíció szerint a $(t, t+h)$ intervallumban a sebesség $v(t)$-ről $v(t+h)$-ra nőtt, azaz az átlaggyorsulás kiszámítható a $\frac{v(t+h)-v(t)}{h}$ differenciahányados segítségével. Amennyiben $h \to 0$-hoz a differenciahányados tart egy értékhez, melyet $v'(t)$-vel jelölünk. Ez a $a(t) = \lim_{h\to 0} \frac{v(t+h)-v(t)}{h}$ mennyiség nem más, mint a $v(t)$ függvény differenciálhányadosa. Vagyis definíció szerint a pillanatnyi gyorsulás a $a(t) = v'(t)$-vel lesz egyenlő. Tehát összefoglalva a fentieket az $a(t) = v'(t) = s''(t)$ összefüggést kapjuk.

A mechanikában a gyorsulás és az erő kapcsolatáról szóló Newton második törvénye kimondja, hogy egy pontszerű test a gyorsulása egyenesen arányos a testre ható erők $\vec{F}$ eredőjével és fordítottan arányos a test m tömegével. Képletben $\vec{F} = m \cdot \vec{a}$. A mozgás leírására keresnünk kell olyan $s(t)$ és $v(t)$ függvénypárt, amelyre $s'(t) = v(t)$ és $v'(t) = \frac{F(s(t))}{m}$. Ennek sok függvénypár tehet eleget, hiszen a mozgó testet sok különböző $s(0)$ helyzetből sok különböző $v(0)$ kezdősebességgel indíthatjuk el. Ha rögzítenénk a $t = 0$ pontban a kezdőhelyzetet és a kezdősebességet, akkor a mozgás, azaz az $s(t)$ függvény már egyértelműen meghatározott lenne. Adott *F*–re az ilyen alakú összefüggést elsőrendű differenciálegyenletnek nevezzük. A differenciálegyenletek olyan egyenletek, melyekben az ismeretlen kifejezés (jelen esetben $s(t)$ függvény) egy differenciálható függvény, és az egyenlet a függvény és annak deriváltja között teremt kapcsolatot. A problémák differenciálegyenletben való megfogalmazása a fizikában, mérnöki tudományokban, a közgazdaságtanban és még számos tudományban alapvető szerepet tölt be.

A differenciálegyenletek az analízisnek egy igen érdekes részét képezik. Speciális esetben léteznek kidolgozott módszerek a megoldásukra, azonban a legtöbb differenciálegyenletet nem tudjuk explicite, képlettel megoldani, hiszen nincsenek általánosan használható megoldási eljárások. Mi most azonban csak közelítő megoldást adunk. A differenciálegyenletek numerikus megoldási módja az hogy, visszavezetjük őket a differenciaegyenletekre, mégpedig úgy, hogy a differenciálegyenletben szereplő differenciálhányadosokat helyettesítjük a megfelelő differenciahányadosokkal.

A differenciálegyenletek visszavezetése ebben a fizikai interpretációban annak felel meg, hogy rögzítünk egy rövid h hosszúságú intervallumot, és egy-egy időközben a mozgást egyenes vonalú egyenletes mozgással közelítjük. Vagyis a vizsgált $[0, t_0]$ időközt bontsuk fel az előre rögzített h hosszúságú részintervallumokra, és ha a j-edik időintervallum a $[t, t+h]$, akkor a $s'(t) = v(t)$ és $v'(t) = \frac{F(s(t))}{m}$ függvénypár helyett használjuk a közelítéseket:

$$\frac{s(t+h)-s(t)}{h}=v(t)$$

$$\frac{v(t+h)-v(t)}{h}=a(t)=F\big(s(t)\big).$$

A test a t pillanatban az $s(t)$ pontba kerül, tehát gyorsulása kiszámítható: $f(s(t))$, sebességét pedig vegyük ismeretlennek. Ennek alapján kiszámolhatjuk a $t+h$-beli sebességet:

$$v(t+h)=v(t)+h\cdot a(t)=v(t)+F(s(t))\cdot h$$

$$s(t+h)=s(t)+v(t)\cdot h$$

Ha jobban megnézzük valójában egy rekurziót írtunk fel a $t+h$-beli útra és sebességre a t-beliekből. Sok fizikai modellezési feladat vezet differenciálegyenletre. Amikor a differenciálegyenleteket közelítjük a differenciaegyenletekkel, tulajdonképpen egy rekurzióval megadott sorozatot kell kiszámolnunk. Egy dologra azonban szeretném felhívni a figyelmet! Amikor egy differenciálegyenletet egy differenciaegyenlettel helyettesítünk, akkor csak közelítő megoldást kapunk. A közelítés annál pontosabb, minél kisebbre választjuk meg a h időközt. Azonban megjegyezném, hogy ha a hatékonyságot is szem előtt tartjuk, akkor célszerű h-t nem túl kicsire választani, mert az nagyon megnövelné a számítási időt.

Egy fizikai mozgás számítógépes feldolgozásakor nyilván nem elméleti tényeket akarunk bebizonyítani. Azonban a számítógép és egy jó program segítségünkre lehet, olyan feladatok modellezésére, amelyeknek tárgyalása nehéz lenne például a középiskolában. Sejtéseink ellenőrzésére nagyszerű ötlet a mozgás pályájának kirajzolása egy-egy konkrét értékre, vagy adott kezdőértékekre a pálya bizonyos adatainak a kiszámítása.

Két időpont között a test mozgását egyenesvonalú egyenletes mozgással közelítjük. A t időpillanatokban vizsgáljuk a mozgó pontot. Kétdimenziós mozgás esetén az x,y koordinátarendszerben vizsgálva a hely, a sebesség és a gyorsulás a tengelyek irányába eső összetevőkre bonthatjuk: a gyorsulás vízszintes összetevőjét jelölje $a_x(t)$, a függőlegest pedig $a_y(t)$. Hasonlóan a sebbeségkoordináták legyenek: a vízszintes összetevő $v_x(t)$ és függőleges $v_y(t)$. A helykoordináták pedig legyenek: $x(t)$ és $y(t)$. A következő rekurziókat használjuk:

- $v_x(t+h)=v_x(t)+a_x(t)\cdot h$
- $v_y(t+h)=v_y(t)+a_y(t)\cdot h$
- $x(t+h)=x(t)+u(t)\cdot h$
- $y(t+h)=y(t)+v(t)\cdot h$

A következő részben egy fizikai modellezésre mutatok példát: a ballisztikus görbét fogom vizsgálni. Célom az, hogy egy program segítségével kirajzoljam egy mozgó test pályáját. Mivel ez már nem része a középiskolás tananyagnak arra szeretnék rámutatni, hogy egy

viszonylag egyszerű program segítségével, hogyan lehetne szemléltetni ennek az anyagnak az elméletét akár a középiskolában is.

A programot QBASIC-ben írtam meg. Azért tértem át elsősorban erre a nyelvre, hiszen most elsődleges célunk a pálya kirajzoltatása lesz. Ebben a környezetben ez könnyen megvalósítható.

Fizikai mozgás: a ballisztikus görbe [1][15][17]

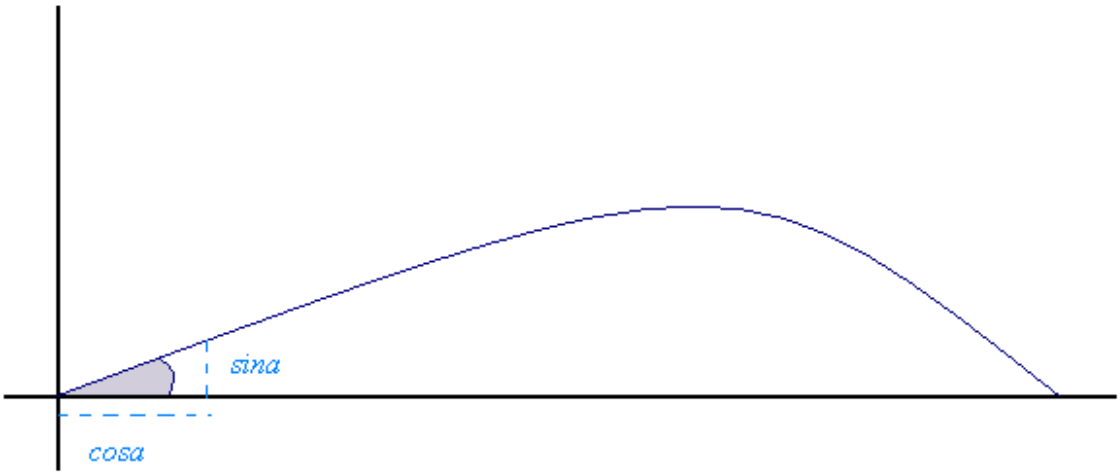
A ballisztika (lövéstan) a lövés jelenségeivel, annak törvényszerűségeivel foglalkozó tudomány. A szó a latin ballista, illetve a görög ballein, hajítani igéből ered. A ballisztika a lövedék mozgása szerint két csoportba osztható: a belső ballisztikára, ami a lövedék mozgását tárgyalja a csőben, illetve a külső ballisztikára, ami a csőből kilépő mozgással, illetve a röppálya leírásával foglalkozik. A matematika és a fizika több kiemelkedő alakja is foglalkozott ezzel a témával. Először 1590 körül Galilei, később Newton, majd Euler dolgozott ki egy közelítő megoldást a ballisztikus pálya leírására. Ezenkívül még Clairaut, Lagrange, Laplace is visszatért erre a tudományterületre.

A dolgozatomban a külső ballisztikával foglalkozom. Ballisztikus görbének a csőből kirepülő lövedék útját nevezzük, ami a cső torkolatától a becsapódásig tart. A pálya üres térben parabola lenne és a gyakorlati számításokban is első megközelítésül parabolának veszik. A kilőtt lövedékre valójában két erő hat. A szabadon eső[1] testek gyorsulását nehézségi gyorsulásnak nevezzük, g-vel jelöljük és az értéke Magyarországon $9{,}81\frac{m}{s^2}$. A nehézségi erő a Föld vonzásának következménye ami fokozatos süllyedésre kényszeríti a lövedéket. A légellenállás olyan közegellenállás, amellyel a mozgó test a levegővel telt térben találkozik. Nagy sebesség esetében ezt leginkább a levegő tehetetlensége okozza. A légellenállás keletkezéséhez a súrlódás is hozzájárul, mely a mozgó test sebességével arányos. A légellenállás a test sebességét csökkenti, ami által folyamatos lassulásra kényszerül a lövedék. Itt felmerül egy probléma, hiszen a légellenállás a magassággal fordítottan arányosan változik. A Föld vonzó erején s a levegő ellenállásán kívül a lövedék kezdősebessége, a levegő sűrűsége illetve a lövedék alakja is közrejátszik a számításokban. A legkisebb légellenállása a csepp alaknak van, ezért nem véletlenül ismerhető fel némi hasonlóság a lövedékek formája és a csepp alakja között.

[1] A testek olyan esését, amely során csak a gravitációs hatás érvényesül (minden más, a mozgást befolyásoló hatás elhanyagolható), szabadesésnek nevezzük.

A ráható erők következtében a test sebessége lecsökken és a lövedék útja egy szabálytalan görbét ír le a levegőben, majd leesik a földre. A lövedék röppályáját két részre oszthatjuk: aktív szakaszra, amikor a ráható erők hatására a lövedék gyorsul, majd a sebesség függőleges összetevője (v_y) megszűnik, illetve a passzívra, amikor a lövedék a tehetetlensége miatt halad tovább. Ahogy az ábrán is megfigyelhető, a leszálló ág rövidebb és íveltebb, mint a felszálló ág, tehát a lövedék mozgásának ideje a röppálya felszálló ágán rövidebb, mint a leszállóágán.

Az is megfigyelhető, hogy a becsapódás szöge, nagyobb, mint a kiinduló szög. A lövedék sebessége kisebb a becsapódó pontban, mint a kezdősebesség. A röppálya alakja attól függ, hogy milyen mértékű az emelkedési szög nagysága, amit jelöljünk α-val. Ha az $\alpha < 35°$, akkor lapos, ha $35° < \alpha < 45°$ közé esik az emelkedési szög, akkor ívelt lesz a röppálya, végül ha $45 > \alpha$, akkor meredek röppályáról beszélhetünk. A legnagyobb lőtávolságot körülbelül 45°-os szögben érhetjük el, ha ennél kisebb vagy nagyobb mértékű szögnél, a röppálya rövidebb lesz, azaz csökken a távolság.



A program

1.Feladat:[1] Egy olyan szimulációs program megírása, ami a légellenállás figyelembe vételével kirajzolja adott értékekre a ballisztikus görbét. Ezzel a példával szeretném

szemléltetni a görbét és jellemzőit. A légellenállás nagy sebességek esetén közelíthető a sebesség négyzetével arányos erővel. Tegyük fel tehát, hogy a légellenállási lassulás (a, b) a sebesség négyzetével, $r^2 = u^2 + v^2$-tel arányos, és ez az arányossági tényező legyen $c = 0{,}00003$. Nézzük a programot:

```
10 SCREEN 12
   PRINT "Ferde hajitas legellenallassal"
   PRINT "idealis esetben a lovedek sebessege: v=1700"
   PRINT "a loves meredesege: m=1"
   PRINT "kozegellanallas: c=0.0003"
30 INPUT "lovedek sebessege"; v
35 IF v = 0 THEN v = 1700
40 INPUT "loves meredksege"; m
45 IF m = 0 THEN m = 1
50 INPUT "kozegellenallas"; c
55 IF c = 0 THEN c = .00003
60 CLS
70 FOR i = 0 TO 600 STEP 3: PSET (i, 320): NEXT i
80 FOR i = 1 TO 300 STEP 2: PSET (5, 320 - i): NEXT i
90 FOR z = 1 TO 1000000: zz = sqrt(2): NEXT z
100 x = 0: y = 0: vx = v * COS(m): vy = v * SIN(m)
110 R = SQR(vx * vx + vy * vy)
120 ax = c * vx * R: ay = c * vy * R
130 vx = vx - ax: vy = vy - ay - 9.81
140 x = x + vx: y = y + vy
145 FOR z = 1 TO 100000: zz = sqrt(2): NEXT z
150 IF 0 > y THEN END
160 IF y > 32000 THEN : GOTO 110
170 PSET (5 + x / 100, 320 - y / 100): GOTO 110
```

A program magyarázata

A sorokat két okból számoztam meg. Első és fő funkciója az volt, hogy a 160-as sorban a `GOTO 110` utasítás értelmezhető legyen. A `GOTO X` utasítás annyit jelent, hogy a program

az x számú vagy címkéjű sornál kezdi a program végrehajtását. Nem törli a gépben már meglévő adatokat (↔ RUN). Jelen programban, ha a 160. sorban teljesül az y>32000 feltétel akkor a program „visszaugrik” a 110. sorba és onnantól fut újra. Második célom pedig a címkézés által a program könnyebb magyarázata. A SCREEN parancs segítségével tudjuk elérni a grafikus üzemmódokat. Minél finomabb a képernyőfelbontás, annál kisebbek a képpontok. Mi a színes SCREEN 12-t használjuk a szép ábrák érdekében. Egy durvább változat is létezik, a SCREEN 13.

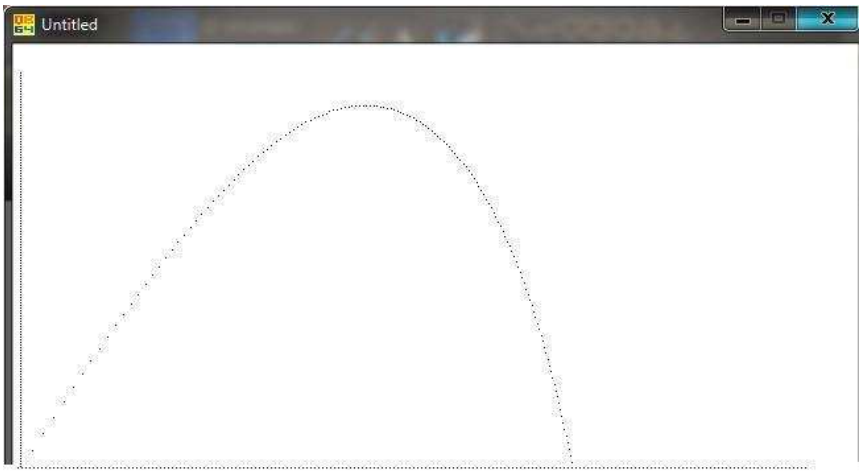
A koordináta-rendszer x tengelye iránya a szokásos, azonban az y tengely felülről lefelé a bal felső sarokból (0,0)-ból a jobb alsó sarokig halad (639,479). Azért van szükség a 70. és a 80. sorokra, hogy ezt a görbe rendszert „helyre állítsuk”. Ha az y tengely állásán változtatunk, akkor természetesen az x-t is hozzá kell igazítani. A koordináta-rendszer tükrözött lesz, mert a bal felső pontban lesz a kiindulópont. A PSET (X,Y)utasítás is a grafika használatához tartozó parancs, segítségével egy pontot tehetünk az (X,Y) helyre. A PRINT kiíratási parancs. Az INPUT után felsorolhatjuk a bekért adatok neveit a ? után. Ez a két utasítás össze is vonható, vagyis PRINT „sebesség=”;: INPUT V helyett INPUT "lovedek sebessege"; v is használható.

Az elágazás szervezés hasonlóan működik mint Pascalban, tehát az IF…THEN…ELSE…(END IF)kulcsszavak segítségével. Ha teljesül az IF utáni feltétel, akkor végrehajtja a THEN utáni utasítást, utasítássorozatot. Ha nem teljesül akkor, az ELSE utániakat. Az END IF-et akkor kell kiírni, ha az IF-THEN-ELSE konstrukció nem egysoros. A 35,45,55. sorokban történik az automatikus értékadás. Ezeknek a szerepe az, hogyha a sebességnek, meredekségnek vagy a közegellenállásnak 0 értéket adunk meg vagy leütjük az entert, akkor az ezekben az elágazásokban szereplő értékeket veszi be a program (alapértékek). CLS a képernyőtörlés felelőse. A 70. és a 80. sor az X és Y tengelyeket nyomtatja ki a képernyőre egy FOR-STEP-NEXT ciklus segítségével, ahol a FOR az alsó érték, TO a felső érték STEP pedig a lépésköz. A FOR-NEXT ciklus ugyanúgy működik, mint az előbbi, csak a lépésközt automatikusan 1-nek veszi.

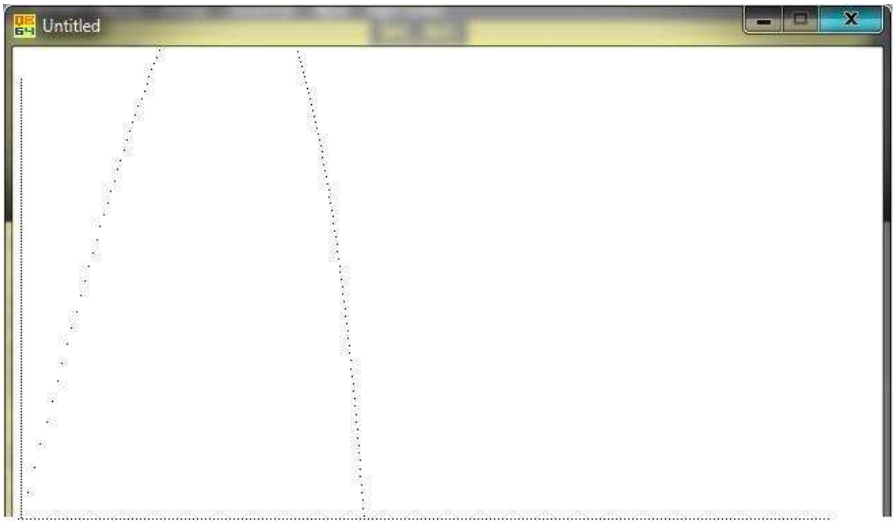
A 100. adja meg a kezdőértékeket, a 110. és a 120. sorok pedig biztosítják, hogy a légellenállás a sebesség négyzetével legyen arányos. A sebességvektort (VX,VY)-vel, a vektor hosszát R-rel jelöli a program. Valóban $(R\cdot VX,R\cdot RY)$ az eredeti (VX,VY)-vel egyirányú lesz és a sebesség négyzetével arányosan hosszú lesz. A 130. sorban a gravitációs

gyorsulás jelenik meg. Probléma lehet, ha a gép nagyon gyorsan nyomtat ki valamit a képernyőre, és nincs idő értelmezni a látottakat. Ennek megoldására szolgál a 90 és 145. sorban található lassító `FOR-NEXT` ciklus. A 150. leállítja a programot, vagyis a lövedék a földbe csapódott. A 170. sor pedig kirajzolja a megadott értékeket.

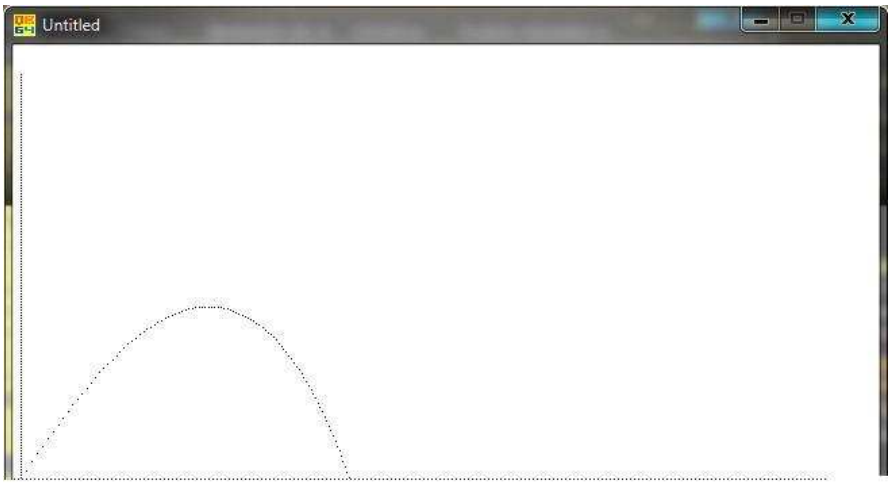
Némi következtetések vonhatók le a következő tesztek eredményeiből. Tekintsük alapadatoknak a $v = 1700,\ m = 1,\ c = 0.0003$ értékeket. Először nézzük meg, hogy ezekkel az adatokkal milyen lesz a kirajzolt ballisztikus görbe! Láthatjuk, hogy valóban egy parabolaszerű görbét rajzol ki, amin a fent említett tulajdonságok jól megfigyelhetők.



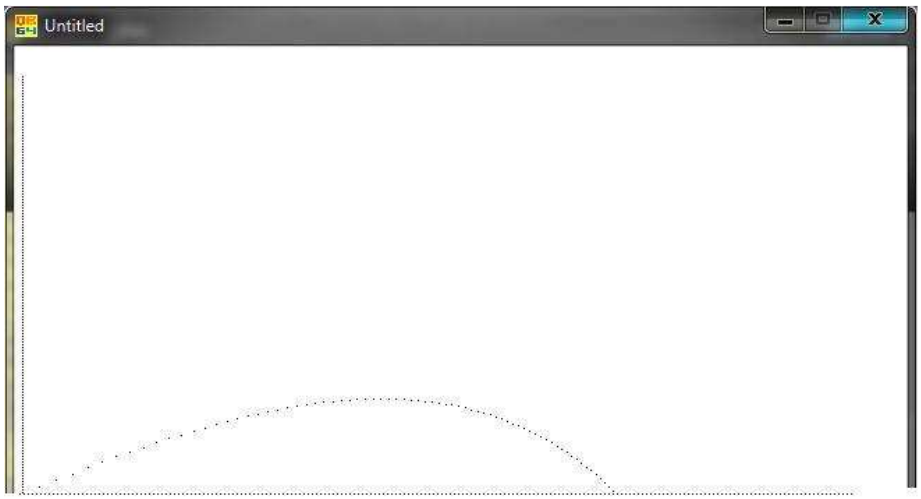
Ha megnöveljük a sebességet ($v = 2000$), akkor a lövedék kimegy a képernyőről. Adatok: $v = 2000, m = 1.3, c = 0.0001$.



Ha lecsökkentjük a sebességet, akkor a lövedék közelebb esik le a kiindulási helyhez képest mint mikor kétszer ennyi sebességgel lőttük ki a lövedéket. Adatok most legyenek: $v = 800, m = 1, c = 0.0001$.



Ha a lövés meredekségét lecsökkentjük a sebességet körülbelül tartjuk és a légellenálláson sem változtatunk, akkor lapos lesz a kirajzolt görbe. Adatok: $v = 1400, m = 0.4, c = 0.0001$.

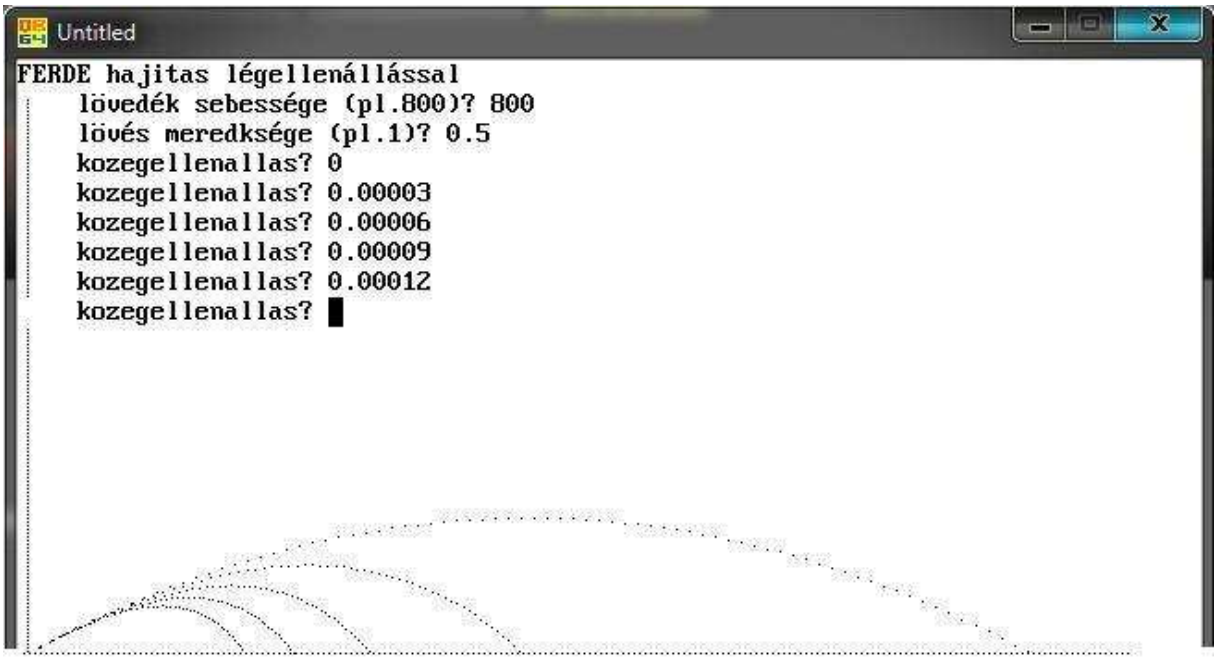


2. Feladat:[1] Módosítsuk a programot úgy, hogy különböző légellenállási értékekre egyetlen koordinátarendszerben rajzolja meg a lövedék pályáját! Mit tapasztalunk? Hogyan befolyásolja a lövés hatósugarát a légellenállás növekedése?

Nézzük a program módosítását:

```
50 INPUT "kozegellenallas"; c
55 IF c = 0 THEN c = .00003
60 CLS
…
150 IF 0 > y THEN END  GOTO 50
```

A `CLS` utasítás kiiktatására azért volt szükség, mert ha a program elér a 150. sorba, majd a `GOTO 50` utasítás miatt visszaugrik az a program 50. sorába, akkor az addig kirajzolt pályák, mindig eltűnnének. A program hibája annyi, hogy sosem áll le, futását megszakítani a *Ctrl* + *Break* billentyűk lenyomásával lehet. A pályákat, amint a képen is látható jól kirajzolja. Levonható következmény, hogy a légellenállás növelésével a lövés hatósugara csökken.



Összegzés

Szakdolgozatom témájául a rekurziókat választottam. Ennek több oka is volt. Amikor a megfelelő témát kerestem, tudtam, hogy az analízis területéről szeretnék választani. A téma konzulense is fontos választási szempont volt nálam. Emellett matematika-informatika szakos tanár hallgató lévén arra törekedtem, hogy munkámban mindkét tudományterület szerephez jusson.

Amit a szakdolgozatomban összefoglaltam a rekurziókról, csupán csak töredéke ennek a témakörnek. Mégis remélem, hogy hasznos dolgokat jártam körül. Az első fejezetben egy rövid összefoglalót írtam a rekurziók elméletéről, néhány tipikus középiskolás feladattal kiegészítve. Majd egy viccre adtam meg a választ, természetesen azt is a „rekurzió nyelvén" megfogalmazva. A második részben a numerikus analízis egy fontos részéről, a Newton-módszerről esik szó. Igyekeztem több színes ábrát is beletenni ebbe a részbe, a módszer gondolatmenetének könnyebb megértése érdekében. Ezt követően két példafeladatot is megoldottam, amihez mellékeltem segédprogramokat. Végül pedig egy fizikai alkalmazást hoztam fel. Itt azért hasznos a mellékelt program, mert ez a témakör nem esik bele a középiskolás tananyagba, és nem is eshetne bele, hiszen a gyerekek nem rendelkeznek megfelelő elméleti hátérrel, hogy ezzel a témakörrel foglakozhassanak. Azonban az elkészített program segítségével, esetleg a fakultáción könnyebb lehetne ennek a témakörnek a megértése. Szakdolgozatom rávilágít arra, hogy a rekurzió alkalmazási területe rendkívül széles, számos tudományágban megtalálható.

Rengeteg kihívást jelentett számomra a dolgozat megírása. Mivel előtte még nem volt részem hasonló feladatban, így a megfelelő téma kiválasztása és leszűkítése hosszú időt igényelt. Az anyaggyűjtés során ügyelnem kellett a megfelelő szakirodalmak kiválasztására, illetve a hibás internetes források felismerésére, majd a különböző jelölésekből egy egységes jelölésrendszer kialakítására. Ennek ellenére nagyon élveztem a kutatást, a programok megvalósítását, illetve a közös munkát a konzulensemmel.

Dolgozatom befejeztével szeretnék köszönetet mondani témavezetőmnek, Gémes Margitnak, aki tudásával és segítőkészségével hozzájárult, mind a szakmai, mind a technikai megvalósításhoz.

Melléklet: CD és tartalma

A szakdolgozatban felhasznált programnyelvek feltelepítése ajánlott, anélkül az alábbi programok ugyanis nem futtathatóak:

- **freepascal.rar -** a Free Pascal v2.24 telepítőfájla

  ingyenesen letölthető: http://www.freepascal.org/

- **comlogo.rar** – a Comenius Logo 3.0 program demo változata[1]

  ingyenesen letölthető: http://comlogo.web.elte.hu/kin001f.htm

- **qbasic.rar -** a QBasic telepítőfájla

  ingyenesen letölthető:http://www.softpedia.com/get/Programming/Coding-languages-Compilers/Qbasic.shtml

A szakdolgozatban szereplő programok:

- **atlos.pas –** kiszámítja egy konvex n-szög átlóinak számát
- **fibonacci.pas** – rekurzívan kiszámítja az n-edik Fibonacci-számot
- **fraktal.lgp –** fraktálrajzoló rekurzív logo program
- **gyokvonas.pas** –négyzetgyökvonó algoritmus
- **newtonmodszer.pas** – egyenlet megoldása a Newton-módszerrel
- **felezosmodszer.pas-** egyenlet megoldásának közelítése
- **pi.pas** – a Newton-módszerrel π közelítő értékének meghatározása
- **ball_gorbe.bas** – adott értékekre a ballisztikus görbe meghatározása
- **legell_gorbe.bas** – a fenti program módosítása: légellenállás növekedése

A szakdolgozat szövege .pdf kiterjesztéssel:

- **Csapo_Zsuzsanna_rekurziok.pdf**

[1] A Comenius Logo 3.0 magyar nyelvű demó verziójában nincs súgó, nem lehet menteni és nyomtatni. Az általam elkészített fraktal.lgp viszont megnyitható és futtatható.

Irodalomjegyzék

[1] **Fried Katalin- Simonovits Miklós:** A problémamegoldás számítógépes iskolája, Typotex Kiadó (Budapest 2005)

[2] **Lothar Berg:** Másodrendű differenciaegyenletek , Tankönyvkiadó (Budapest, 1982)

[3] Matek Portál: http://matek.fazekas.hu/portal/tanitasianyagok/Orosz_Gyula/Rek/index.htm

[4] **Dr. Szőnyi Tamás:** Véges matematika 2 jegyzete

[5] Wikipédia: http://hu.wikipedia.org/wiki/Newton-m%C3%B3dszer

[6] **Dr. Máté László:** Rekurzív sorozatok, Tankönyvkiadó (Budapest, 1980)

[7] **Angster Erzsébet:** Programozás tankönyv I.- Strukturált tervezés Turbo Pascal, Akadémiai Nyomda (Martonvásár, 1998)

[8] **Papp- Varga Zsuzsanna:** Programozás nyelvi eszközei az oktatásban jegyzete

[9] **Dr. Krebsz Anna**: Numerikus analízis jegyzete

[10] www.inf.u-**szeged**.hu/~verkri/munka/**newton_modszerek**.pdf

[11] **George B. Thomas, Maurice D. Weir, Joel Hass, Frank R. Giordano:** Thomas-féle kalkulus I., Typotex kiadó (Budapest, 2008)

[12] bme.ysolt.net/2_felev/Matek_A2/Kapott/Mate.../B17_tetel.pdf

[13] **Stoyan Gisbert, Takó Galina:** Numerikus módszerek I., Typotex kiadó (Budapest, 1993)

[14] **Dr Sikolya Eszter:** Analízis 2 jegyzete

[15] fegyvermester.hu/fegyverism/**11_ballisztika**.pdf

[16] **Laczkovich Miklós-T.Sós Vera:** Analízis I., Nemzeti Tankönyvkiadó (Budapest, 2005)

[17] **dr. Halász Tibor:** Fizika 9., Mozaik Kiadó (Szeged, 2006)

[18] **Benkő Tiborné, Tóth Bertalan:** Együtt könnyebb a programozás: Free Pascal, Computerbooks (Budapest,2010)

[19] **Lovász László, Pelikán József, Vesztergombi Katalin:** Diszkrét matematika, Typotex Kiadó (Budapest 2010)