EGYVÁLTOZÓS FÜGGVÉNYEK ÁBRÁZOLÁSA

Szlávi Péter

2000-2014

TARTALOM

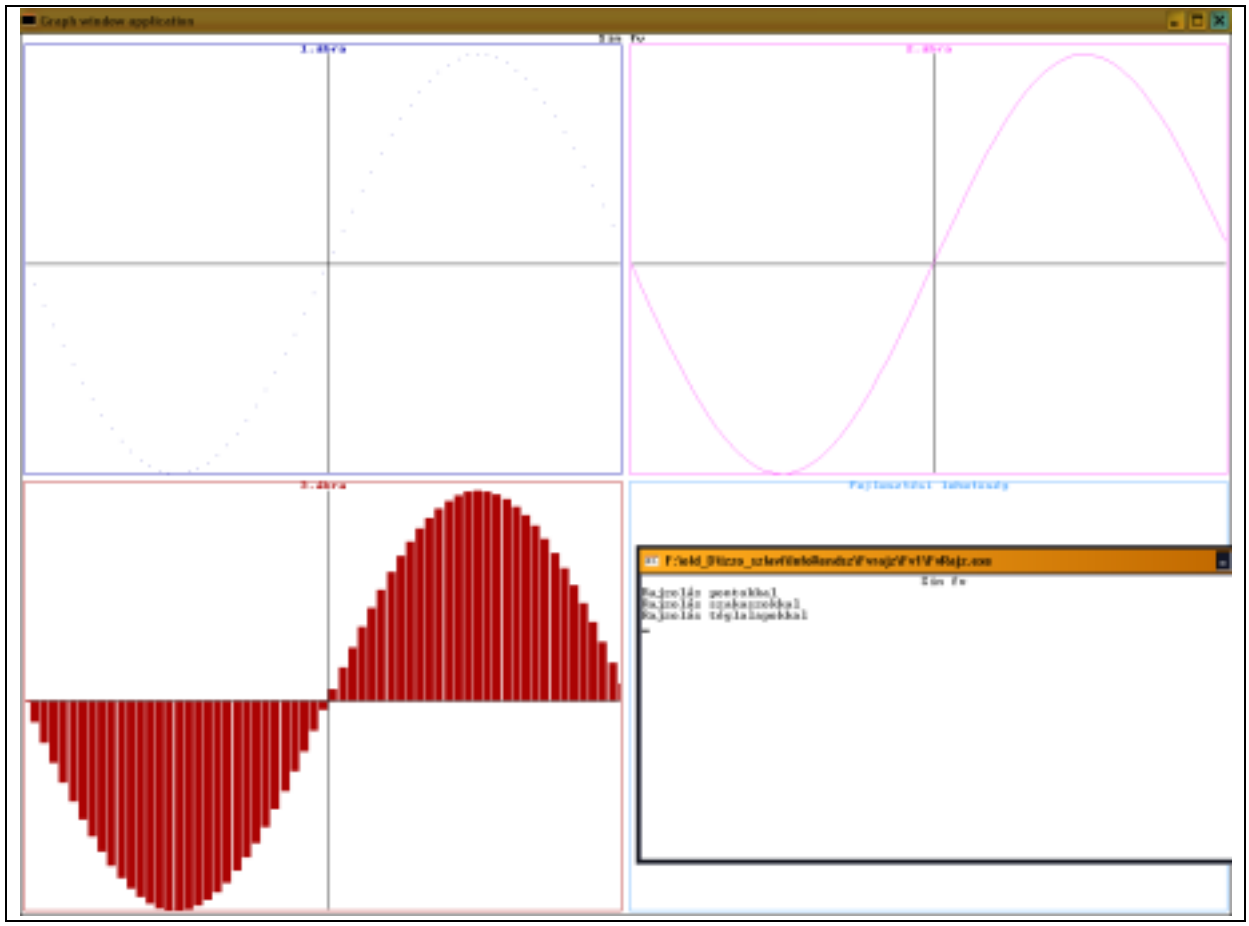
Egyváltozós függvények ábrázolása 1
1 Bevezetés 2
2 Útban a megoldás felé 2
2.1 Jelölések 2
2.2 Problémák 3
2.3 A problémák megoldása 3
3 Módszerek – algoritmusok 4
3.1 Bamba, de egyszerű… 5
3.2 Képernyőre normálva 5
3.3 Aránytartva normálva 6
3.4 Folytonos vonallal 6
3.5 Téglákkal 6
3.6 Trapézokkal 6
3.7 Görbeívekkel 7
4 A keretprogram 9
5 A megoldás 10

1 Bevezetés

Feladat egy egyváltozós valós függvény kirajzolása különféle megjelenítési módszerekkel. Például:

- pontokkal,
- folytonos szakaszokkal,
- téglalapokkal,
- … .

Az elvárásokat legjobban az alábbi, futás során keletkezett ábrasor fejezi ki,



1. ábra. Egy futási kép – a sinus függvény.
(FP-ben két ablak: az egyik a vezérléshez, a másik a grafikus megjelenítéshez kell.)

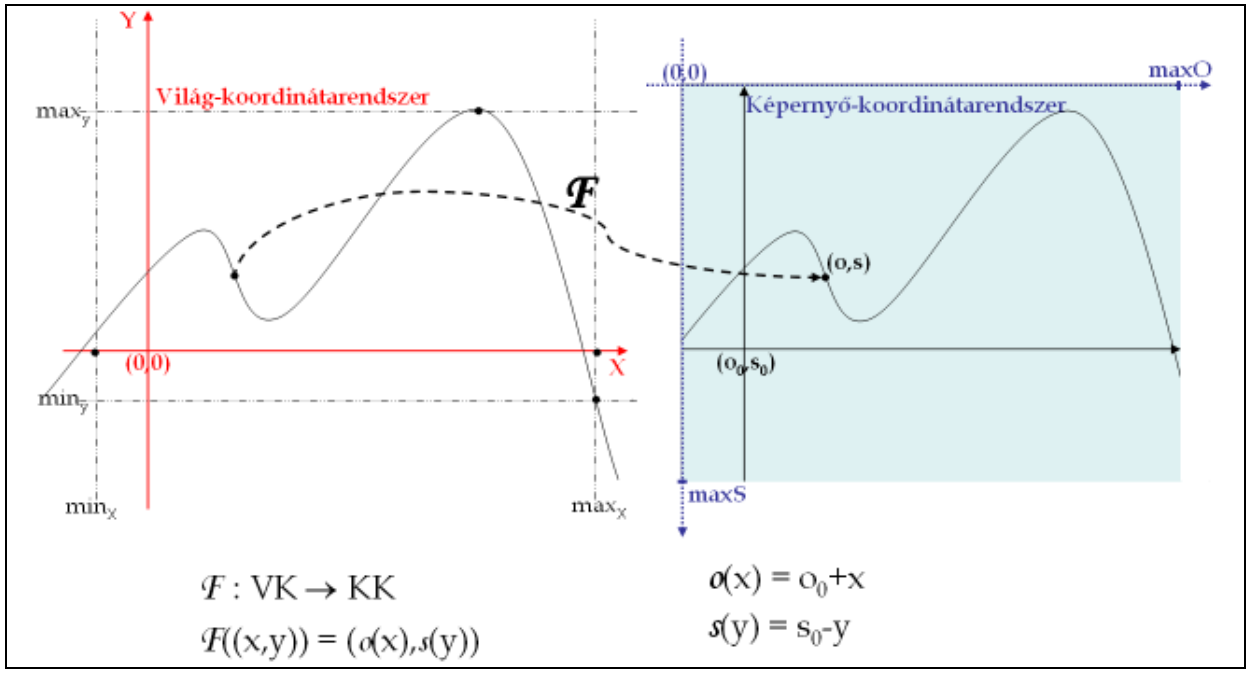
valamint egy (teljesebb) próba: Fv1Rajz.exe.

2 Útban a megoldás felé

2.1 Jelölések

Alapadatok:

- f – függvénykapcsolat; $f: \mathcal{D}_f \rightarrow \mathcal{R}_f$,
- $\mathcal{D}_f$ – (az „érdekes”) értelmezési tartomány = $[\min_x..\max_x]$
- $\mathcal{R}_f$ – (az „érdekes”) értékkészlet = $[\min_y..\max_y]$



2. ábra. A transzformáció „ránézésre”.
(VK = világ-koordinátarendszer, KK = képernyő-koordinátarendszer)

A (lineáris) transzformáció ellenőrzése 2, különböző pontra:

- Az *origó* leképezése: $\mathcal{F}((0,0)) = (o_0+0, s_0-0) = (o_0, s_0)$
- A *bal-felső sarok* leképezése: $\mathcal{F}((-o_0, s_0)) = (o_0-o_0, s_0-s_0) = (0,0)$

Megjegyzés: a **képernyő** helyett mondhatnánk a képernyőn megjelenő **ablak**ot[1] is.

2.2 Problémák

1. túl szűk/tág az értékkészlete (pl. sin(-π..π) / exp(0..100))
2. túl szűk/tág az értelmezési tartománya (pl. sin(-π..π) / 1/x (x=1..1000))
3. függőlegesen alul/felül kilóg (pl. $-10-x^2$ (x=-100..100) / $10+x^2$ (x=-100..100))
4. vízszintesen balra/jobbra kilóg (pl. $(100+x)^2-10$ (x=10..100) / $(100+x)^2-10$ (x=-100..-10))

2.3 A problémák megoldása

Léptékezés vagy –másként– **nyújtás** (1-2.-re), az origó **eltolás**a (3-4.-re).

Nyújtás (a teljes [0..maxO]×[0..maxS] képernyőre)

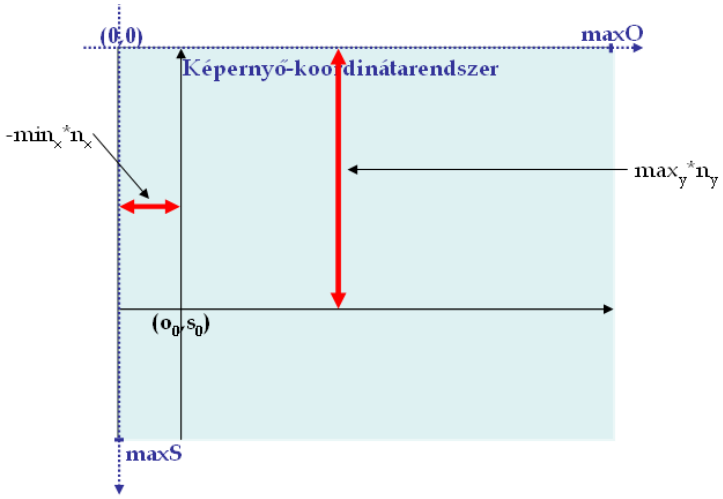
- X-irányban: $n_x = (maxO+1) / (max_x - min_x)$
- Y-irányban: $n_y = (maxS+1) / (max_y - min_y)$

A számlálóbeli +1 a képernyő-koordinátarendszer egész léptékének köszönhető: annyi darab pont van az oszlopokban, illetve a sorokban.

[1] A pontosabb szóhasználat a VK-ban beszél **ablak**ról, a KK-ban **képernyőtartomány**ról (rövidebben: **képtartomány**ról).

Origóeltolás

- $o_0 = -min_x * n_x$
- $s_0 = max_y * n_y$
  $\equiv maxS + min_y * n_y$



Így a keresett $\mathcal{F}$ **leképezés**:

- $o(x) = o_0 + x * n_x$
- $s(y) = s_0 - y * n_y$

3 Módszerek – algoritmusok

Az algoritmus ötlete pofonegyszerű:

1. **generáljuk le** az ábrázolandó függvényt egy ún. **függvénytáblába**, valamekkora lépésközzel, célszerűen: ekvidisztáns alappontok felett, növekvő x-ek mentén;
2. majd (most már függvény kiszámítási szabályától függetlenül) **rajzoljuk ki** a függvénytáblában lévő pontokon nyugvó függvénygörbét, valamilyen módszerrel.

Algoritmikus adatok és egyéb kellékek:

```
Függvény f(Konstans x:Valós):Valós
  … a kirajzolandó függvényt leíró függvényeljárás …

Típus
  TKoordináták    = Tömb(1..MaxN:Valós) [koordináták]
  TFüggvényTábla = Rekord(
                      n:Egész,
                      x,y:TKoordináták [2],
                      minY,maxY:Valós)

Változó [globálisak = a feladat meghatározta paraméterek]
  ft:TFüggvényTábla              [egy fv. gráfjának pontjai]
  mag,szél:Egész                 [az aktuális képernyő/ablak méretei]
```

A fentieket elegendő egyszer meghatározni. Minden függvényrajzoló módszer használja.

```
Változó [az egyes módszerekhez, lokálisak]
  o0,s0:Egész                    [az origó helye a képernyőn/ablakban]
  nX,nY:Valós                    [nyújtási tényezők]
  dX:Egész                       [oszlopszélesség]
  eO,eS:Egész                    [az előzőleg kirajzolt pont]
```

A fentieket elegendő módszerenként egyszer meghatározni. Ezeket is, mint „látens” (azaz explicit átadás nélküli) paramétereket, használhatják a módszerek.

[2] Az x-koordináták növekvően rendezettek ⇒ `minX=ft.x(1), maxX=ft.x(ft.n)`.

```
Eljárás PontRajzolás(Konstans x,y:Valós):
  [Ef: helyes o0,s0, nX,nY]
  Változó
    o,s:Egész
  o:=o0+Kerekít(x*nX); s:=s0-Kerekít(y*nY)
  Pont(o,s)
  [eO:=o; eS:=s] [3]
Eljárás vége.
```

[A Pont eljárás ún. grafikus primitív, Turbo Grafikában [4]: **PutPixel**.]

```
Eljárás SzakaszRajzolás(Konstans hozX,hozY:Valós):
  [Ef: az (eO,eS) pont már ki van rajzolva, s ez az aktuális]
  Változó
    s,o:Egész
  o:=o0+Kerekít(hozX*nX); s:=s0-Kerekít(hozY*nY)
  Szakasz(eO,eS,o,s)
  eO:=o; eS:=s
Eljárás vége.
```

[A Szakasz eljárás ún. grafikus primitív, TG-ben: **Line**.]

```
Eljárás TéglaRajzolás(Konstans x,y:Valós):
  [Ef: dX=az oszlopok szélessége (Egész)]
  Változó
    s,o:Egész
  o:=o0+Kerekít(x*nX); s:=s0-Kerekít(y*nY)
  Tégla(o,s,o+dX,o0-sgn(y) [ne érjen rá az x-tengelyre! [5]] )
Eljárás vége.
```

[A Tégla eljárás ún. grafikus primitív, TG-ben: **Bar**.]

3.1 Bamba, de egyszerű…

Nem figyeljük a „képernyőre miként kerülést”… az origó középen, léptékezés nincs. A „rá-nézésre” transzformáció méltó párja:

```
…
o0:=maxO Div 2; s0:=maxS Div 2; nX:=1,0; nY:=1,0
Ciklus i=1-től ft.n-ig
  PontRajzolás(ft.x(i),ft.y(i))
Ciklus vége
…
```

3.2 Képernyőre normálva

```
[ef: ft.x szigorúan növekvően rendezett tömb ⇒
          ∀i∈[1..ft.n): ft.x(i)<ft.x(i+1)]

…
nX:=(maxO+1)/(ft.x(ft.n)-ft.x(1)); nY:=(maxS+1)/(ft.maxY-ft.minY)
o0:=-nX*ft.x(1); s0:=maxS+nY*minY
Ciklus i=1-től ft.n-ig
  PontRajzolás(ft.x(i),ft.y(i))
Ciklus vége
…
```

---

[3] A `SzakaszRajzolás` eljárás majd épít arra, hogy az utoljára kirajzolt pont helye meglegyen az `(eO,eS)`-ben.
[4] Akár Turbo Pascalban, akár Free Pascalban a `Graph` unit tartalmazza a grafikus „szókincset”.
[5] Meggondolandó az x=0, és az y=0 esete!

A **konstans függvény**re ez rossz eredményt ad, ui. az nY kiszámolása közben a nevezőbe 0 kerül. Ez elkerülhető, ha erre előre gondolunk a függvénytábla generálásakor. Ilyen esetben az alábbi módosítás megfelelő, ugyanis így még az x-tengely is a képre kerül:

```
Ha ft.minY=ft.maxY akkor
  k:=ft.minY
  Elágazás
    k>0 esetén ft.maxY:=3*k; ft.minY:=-k
    k<0 esetén ft.maxY:=-k;  ft.minY:=3*k
    k=0 esetén ft.maxY:=+1;  ft.minY:=-1
  Elágazás vége
Elágazás vége
```

A **konstans függvény** e fajta kezelését a későbbiekben is feltesszük.

3.3 Arányokat tartva normálni

Ha függvény lényegéhez tartozik az x-y aránya, akkor a fenti rajzolási módszer nem alkalmazható.

```
[ef: ft.x szigorúan növekvően rendezett tömb]

…
nX:=(maxO+1)/(ft.x(ft.n)-ft.x(1)); nY:=(maxS+1)/(ft.maxY-ft.minY)
Ha nY>nX akkor nY:=nX különben nX:=nY [egyszerűbben: nX,nY:=Min(nX,nY)]
o0:=-nX*ft.x(1); s0:=maxS+nY*minY
Ciklus i=1-től ft.n-ig
  PontRajzolás(ft.x(i),ft.y(i))
Ciklus vége
…
```

Egy jövőbe mutató extra probléma: hogyan lehetne megvalósítani egy ábrán több függvény kirajzolását, ha ügyelni kell az arányokra (az egymáshoz s esetleg: saját magukhoz)? Fölteheti, hogy a függvények értelmezési tartománya ugyanaz, sőt az alappontok is ugyanazok.

3.4 Folytonos vonallal

```
[ef: ft.x szigorúan növekvően rendezett tömb]

… [a 3.2 vagy a 3.3 inicializálása] …
PontRajzolás(ft.x(1),ft.y(1))
Ciklus i=2-től ft.n-ig
  SzakaszRajzolás(fv.x(i),ft.y(i))
Ciklus vége
…
```

3.5 Téglákkal

```
[ef: ft.x szigorúan növekvően rendezett, és azonos különbségű tömb]

… [a 3.2 vagy a 3.3 inicializálása] …
Ciklus i=1-től ft.n-ig
  TéglaRajzolás(ft.x(i),ft.y(i))
Ciklus vége
…
```

3.6 Trapézokkal

Ez ötvözi a szakaszokkal és a téglákkal történő rajzolást. Hátránya, általában kevesebb segítséget nyújtanak a programnyelvek. [6] Elemibb műveletekkel (szakaszrajzolás és tartományszínezés) persze nem nagy munka árán megoldható.

---

[6] Üdítő kivétel a Turbo Grafika e célra kiváló két eljárása:

Az algoritmizálás és kódolás: hf.

3.7 Görbeívekkel

Számos módszer közül választhatunk. Legkézenfekvőbb, hogy az N pontra ráfektethetünk egy N-1-ed fokú polinomot. Másik szokásos módszer (ill. inkább módszercsalád), hogy a szomszédos pontokra valahányadfokú (de egyedenként paraméterezett) polinomot illesztünk úgy, hogy az adott pontokon átmenjenek, sőt –hogy ez az „átmenet" az egyik polinomból a másikba minél „simább" legyen– az érintők egyenlőségét is meg szokták követelni.

Az algoritmizálás és kódolás: hf.

Segítségképpen alább egy könnyen megvalósítható módszerről ötletelek:

1. Az 1-3.pontra ráfektetünk egy parabolaívet. (Ez egyértelműen elvégezhető.)
2. A 3.-kal kezdve az egymást követő pontpárokra úgy fektetünk parabolaíveket, hogy az a bal oldali szomszédjához „érintőlegesen" is illeszkedjen.
3. A parabolaíveket $f_i(x)=a_ix^2+b_ix+c$ alakban írjuk föl. Az A$\underline{\mathbf{x}}$=$\underline{\mathbf{b}}$ vektoregyenletet kell megoldanunk úgy, hogy az A együtthatómátrixot tartalmazza azon lineáris egyenletek alappontok x-eiből számított konstansait, amelyek az 1-2. alapján készültek. Az A$\underline{\mathbf{x}}$=$\underline{\mathbf{b}}$ egyenlet túl oldali vektorának elemei ($\underline{\mathbf{b}}$) részben az y-okból fog állni, részben egyéb megfontolásokból származnak.
4. Képezzük az együtthatómátrix inverzét (A^{-1}).
5. Megoldjuk az egyenletet: $\underline{\mathbf{x}}=A^{-1}\underline{\mathbf{b}}$.

Megjegyzem: van számos hatékony módszer a numerikus matematikában lineáris egyenletrendszerek megoldására; így a mátrix invertálás valójában nem az egyedül üdvözítő módszer. Ilyen módszer például a Gauss-Jordan elimináció. (Ez az $\underline{\mathbf{x}}$ vektor mellett az A^{-1}-et is előállítja, bár minket csak az $\underline{\mathbf{x}}$ érdekel.) [7]

Nézzünk egy konkrét példát!

Ha 4 pontra kell fektetni görbeíveket, akkor 2 parabolaívvel oldható meg a probléma. Az első 3 pontra egyértelműen fektethetünk egyet, majd a 3. és 4. pontra azzal a plusz feltétellel, hogy ez utóbbi érintője a 3. pontban egyezzen meg az első ugyanezen pontbeli érintőjével. Azaz a keresett 6 ismeretlen paraméterre az alábbi egyenleteket állíthatjuk föl.

$f_1(x_1)=y_1$, $f_1(x_2)=y_2$, $f_1(x_3)=y_3$
$f_2(x_3)=y_3$, $f_2(x_4)=y_4$, $f_1'(x_3)-f_2'(x_3)=0$

A részletszámítások nélkülözésével, csak az egyes lépések végeredményét mutatja az alábbi ábra.

```
Procedure DrawPoly(Const db:Word; Const pontok);
Procedure FillPoly(Const db:Word; Const pontok);
 {a pontok: db darab, a Graph unitban definiált PointType típusú adat kezdőcíme}
Type PointType=Record x,y:Integer End;
```

[7] Ennek kipróbálására szíves figyelmükbe ajánlom:
GaussJordan.exe programot, illetve a forrását: GaussJordan.pas!

i:	1	2	3	4
x_i:	0	1	2	3
y_i:	1	4	-2	4

b	Mátrix (A):					
1	0	0	1	0	0	0
4	1	1	1	0	0	0
-2	4	2	1	0	0	0
-2	0	0	0	4	2	1
4	0	0	0	9	3	1
0	4	1	0	-4	-1	0
	a_1	b_1	c_1	a_2	b_2	c_2
x	Inverz mátrix (A^{-1}):					
-4,5	1	-1	1	0	0	0
7,5	-2	2	-1	0	0	0
1	1	0	0	0	0	0
16,5	-1	2	-2	-1	1	1
-76,5	3	-10	8	4	-4	-5
85	-3	12	-9	-3	4	6

Egyenletek:

$f_i(x)=a_i x^2+b_i x+c_i$

$\leftarrow f_1(x_1)=y_1$

$\leftarrow f_1(x_2)=y_2$

$\leftarrow f_1(x_3)=y_3$

$\leftarrow f_2(x_3)=y_3$

$\leftarrow f_2(x_4)=y_4$

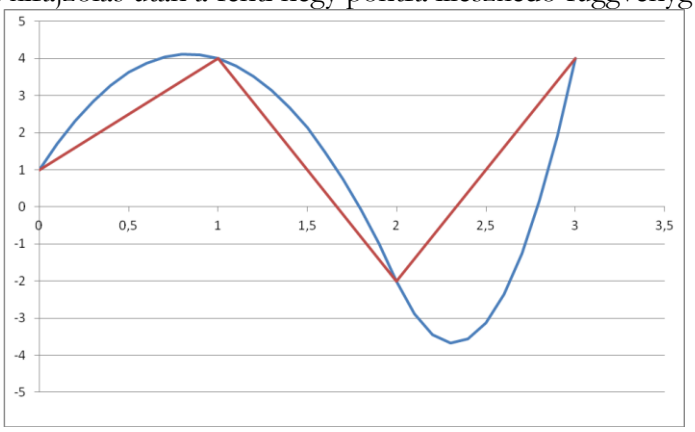
$\leftarrow f_1'(x_3)-f_2'(x_3)=0$

Tehát megkaptuk a két parabolaív paramétereit:

$$f_1(x)=\mathbf{-4,5}x^2+\mathbf{7,5}x+\mathbf{1} \quad x\in[x_1..x_3],\ f_2(x)=\mathbf{16,5}x^2-\mathbf{76,5}x+\mathbf{85} \quad x\in[x_3..x_4]$$

Hogy érjünk is ezzel az izzasztó eredménnyel valamit, az adott x-pontok között finomabb lépésekben is számíthatjuk az interpolált értékeket. Ezáltal kevés pont esetén is „szép” görbével tudjuk megjeleníteni az adott görbeívet.

Lássuk, mit kapunk a kirajzolás után a fenti négy pontra illeszkedő függvénygörbe gyanánt!



Ahhoz képest, hogy eredetileg összesen csak négy pont volt adva (pirossal szakaszokkal jelölve), elégedettek lehetünk a látvánnyal (kékkel jelölve). Itt összesen 8 köztes pontot számoltunk az egyes rögzített pontok között.

Az itt körvonalazott módszer (kirajzolás nélküli, a keretbe egy az egyben *nem* beilleszthető) megvalósítását is megtalálhatja: GaussJordan.zip[8].

Nézzünk egy másik –talán természetesebbnek ható –módszert!

Tegyük ismét föl, hogy 4 pontra kell fektetni görbeíveket. Most 3 parabolaívvel oldjuk meg a problémát, de „egységesebb” gondolattal vezérelve. Minden egymást követő pontpárra fektetünk parabolát, természetesen úgy, hogy illeszkedjenek a pontokra, és „érintőlegesen” simuljon a bal oldali szomszéd parabolához. 4 pont esetén 3 parabola, ami 9 ismeretlent jelent. Az alábbi egyenleteket állíthatjuk föl.

[8] A tömörített állományban egyrészt az a program (pas, exe) található, amely konstans formában megadott pontokhoz meghatározza a fentebb vázolt parabolaívek paramétereit. Mellesleg tartalmazza azt a programot, amely demonstrálja –feladat-független módon– a lineáris egyenletrendszert megoldó Gauss-Jordan módszer algoritmusát.

$f_1(x_1)=y_1$, $f_1(x_2)=y_2$, $f_2(x_2)=y_2$, $f_2(x_3)=y_3$, $f_3(x_3)=y_3$, $f_3(x_4)=y_4$,
$f_1'(x_2)-f_2'(x_2)=0$, $f_2'(x_3)-f_3'(x_3)=0$

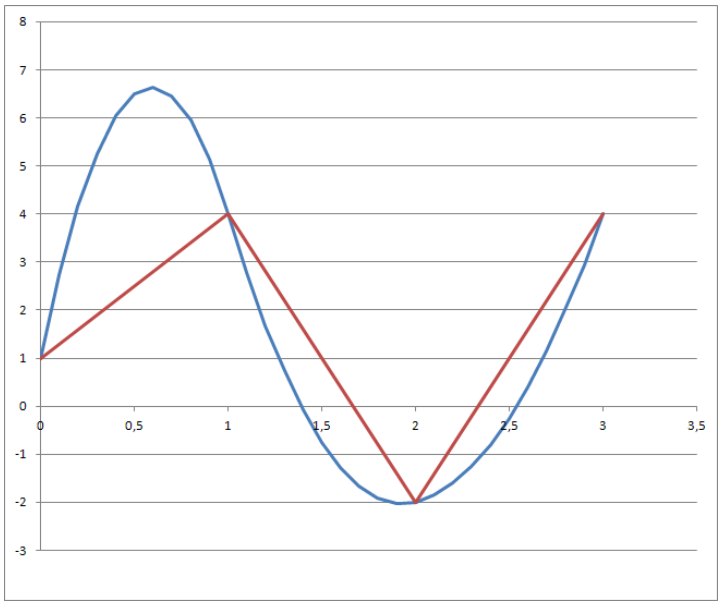
Ez azonban még nem elég a megoldhatósághoz! Kell még találnunk egy kielégítendő –fentiektől független– „szabályt”, egyenletet! Legyen ez pl. a következő: az első parabolaív érintője a bal oldali pontjánál, az utolsó parabolaív érintője a jobb oldali pontjánál legyen egymásra „tükrös”, azaz

$f_1'(x_1)+f_3'(x_4)=0$

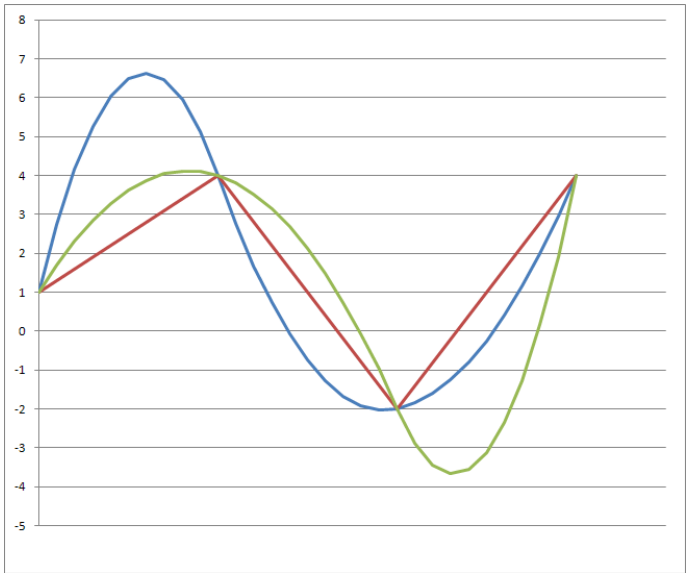
Lássuk, mit kapnánk az egyenletrendszer megoldása után!

$f_1(x)=-16x^2+19x+1 \quad x\in[x_1..x_2]$,
$f_2(x)=7x^2-27x+24 \quad x\in[x_2..x_3]$,
$f_3(x)=5x^2-19x+16 \quad x\in[x_3..x_4]$.

A fenti négy pontra illeszkedő függvénygörbe most így fest!



Érdemes „egymásmellé” tenni az összevetés kedvéért, és következtetéseket le vonni!



4 A keretprogram

Lásd `Fv1RajzK.pas`.

5 A megoldás

A 3 tervezett és fentebb kidolgozott megoldást tartalmazzák az alábbi fájlok: `Fv1Rajz.pas`.

Házi feladatként érdemes a 3.6-ban vagy a 3.7-ben körvonalazott megjelenítési módokat megvalósítani. A 3.6-hoz lényegileg minden ismert (az anyag alapján). A 3.7-ben körvonalazott módszer (a hivatkozott programban megvalósított kóddarab felhasználásával) már viszonylag könnyen elkészíthető. „Csupán" az adott szomszédos pontokat összekötő parabolaíveket finomabb lépésközű „alpontokra" kell illeszteni, (pontokkal, szakaszokkal vagy téglákkal) kirajzolni.