3-ad rendű fapados bezier

```
x[0] := 14;
y[0] := 1
x[1] := 28
y[1] := 48
x[2] := 50
y[2] := 41
x[3] := 40
y[3] := 25

#alappontok -> 3-ad rendű polinom illesztése
```

```
#1 fapados
Bezier := proc(x, y)
    local f, g;    #paraméterek
    local c;       #plot
    f := t→(x[0]·t^3) + (3·x[1]·t^2·(1 − t)) + (3·x[2]·t·(1 − t)^2) + (x[3]·(1 − t)^3);
    g := t→(y[0]·t^3) + (3·y[1]·t^2·(1 − t)) + (3·y[2]·t·(1 − t)^2) + (y[3]·(1 − t)^3);
    c := plot([f(t), g(t), t = 0 ..1], thickness = 2);
    display(c);
end proc

> Bezier(x, y)
```

```
with(plots)
```

3-ad rendű fullos bezier

```
#a fenti megoldást egészitsük ki 2 dologgal
#1: pontok névvel együtt
#2: konvex burok
BezierFull := proc(x, y)
    local f, g;             #paraméterek
    local c;                #plot
    local p0, p1, p2, p3;   #szövegek
    local l1, l2, l3;       #vonalak
    local pp;               #pontok
    f := t→(x[0]·t^3) + (3·x[1]·t^2·(1 − t)) + (3·x[2]·t·(1 − t)^2) + (x[3]·(1 − t)^3);
    g := t→(y[0]·t^3) + (3·y[1]·t^2·(1 − t)) + (3·y[2]·t·(1 − t)^2) + (y[3]·(1 − t)^3);
    p0 := textplot([15, 5,'P0'], align = ABOVE);
    p1 := textplot([29, 48,'P1'], align = RIGHT);
    p2 := textplot([49, 42,'P2'], align = LEFT);
    p3 := textplot([44, 25,'P3'], align = ABOVE);
    l1 := line([ x[0], y[0] ],   [x[1], y[1]  ], color = blue);
    l2 := line([ x[1], y[1] ],   [x[2], y[2]  ], color = blue);
    l3 := line([ x[2], y[2] ],   [x[3], y[3]  ], color = blue);
    pp := pointplot({  [x[0], y[0]],   [x[1], y[1]],   [x[2], y[2]],   [x[3], y[3]   ]}, symbol = circle);

    c := plot([f(t), g(t), t = 0 ..1], thickness = 2);
    display(pp, c, p0, p1, p2, p3, l1, l2, l3);
end proc
> BezierFull(x, y)
```

```
with(plottools)
```

4-ed fokú bezier képlet:

$$f := t \to \left(x[0]\cdot t^4\right) + \left(4\cdot x[1]\cdot t^3\cdot(1-t)\right) + \left(4\cdot x[2]\cdot t^2\cdot(1-t)^2\right) + \left(4\cdot x[3]\cdot t\cdot(1-t)^3\right) + \left(x[4]\cdot(1-t)^4\right);$$

Függvények:

2 módszer:

$f := 2\cdot\sin(3\cdot x - Pi) + 1$

$g := x \to 0.5\cdot\cos(2\cdot x + 0.5\cdot Pi) - 2$

#helyettesítési érték

evalf(subs(x = 2, f))

evalf(g(2))

Darabonként összerakott fgv:

$signum1 := x \to piecewise(x < 0, -1, x = 0, 0, x > 0, 1)$

Plot konfig: with(plots)

> plot(signum1, −50 ..50, color = "red", labels = ["x-tengely", "y-tengely"], title = "Szignum fgv", gridlines, view = [−30 ..30, −2 ..2], linestyle = dashdot)

Viszgálat (sima fgv!!):

folytonosság: #1 Folytonosság szakadási hely

#iscont library segítségével

readlib(iscont)

proc

> $fgv2 := \frac{x^2}{(x-1)\cdot(x+2)}$

> plot(fgv2)

iscont(fgv2, x = 1 ..5) #folytonos

iscont(fgv2, x = 0 ..5) #nem folytonos

discont(fgv2, x)

---

határérték:

> #2 fgv határérték

> #limit függvénnyel

#limit(fgv neve, hely, határérték típusa)

> limit(fgv2, x = 1, right)

> limit(fgv2, x = 1, left)

> $limit\left(\frac{1}{x}, x = 0, left\right)$

> $limit\left(\frac{1}{x}, x = 5\right)$

szélsőérték:

> #3 szélsőérték

#globális-lokális minimum-maximum

#maximize, minimize(fgv, változó neve)

> maximize(fgv2, x)

> minimize(fgv2, x)

> maximize(fgv2, x = −20 .. −10)

> minimize(sin(x), x)

Deriválás:

- #DERIVÁLÁS
- #függvény deriváltja: adott pontban húzott érintőegyenes meredeksége
- #mire használjuk?
  - szélsőérték megoldás,
  - $\frac{\text{érintőegyenes egyenletének}}{\text{meredekségének}}$ felírása
  - integrálás

```
f2 := x·y + x·cos(3·y + x)

diff(f2, x)      df2 := diff(f2, x)

diff(f2, y)      evalf(subs(x = 1, df2))
```

#magasabb rendű deriválás

```
diff(f2,  x, y, y)
```

Taylor: #taylor polinommal számított közelítő érték
#nem lineáris függvény közelíthető egy polinommal

> #taylor sor

$$f3 := \frac{(e^x - 1)}{x}$$

$$t3 := \text{taylor}(f3, x = 1, 10)$$

$$p3 := \text{convert}(t3, \text{polynom})$$

#taylor(fgv, változó neve és x0 kezdőpont, polinom fokszáma)

#a kapott p3 polynom mennyire jól közelíti az eredeti f3 fgv-t?
#egy adott pontban számítsuk mindkettő behelyettesítését

$$\text{evalf}\left(\text{subs}\left(x = \frac{Pi}{3}, f3\right)\right) \qquad -0.5000000005$$

$$\text{evalf}\left(\text{subs}\left(x = \frac{Pi}{3}, t3\right)\right) \qquad -0.4995669892 + O\left(\frac{1}{59049}\pi^{10}\right)$$

Integrálás:

#ha létezik primitív fgv: egzakt módon integrálható + határozott integrál számítható
#ha nem létezik primitív fgv: közelítő integrál: (mindig határozott integrált számítunk)
– simpson
– trapéz
– stb.

#library: student

```
readlib(student)
with(student)
```

#1.van primitív fgv.

$$f1 := x^2 + 2\cdot x + \frac{1}{x}$$

```
int(f1, x)
```

#ln(x) az abszolútértékben!

#határozott integrál:

```
evalf(int(f1, x = 3..7))
```

#parciális törtekre bontás

$$f2 := \frac{x}{(x+1)\cdot(x-2)}$$

```
int(f2, x)
```

> #parciális integrálás (szorzatra)

$$f3 := x\cdot\sin(3\cdot x)$$

```
int(f4, x = 1..3)
```

#2.Nincs primitív függvény

> $f1 := \frac{\exp(x) - 1}{x}$

> $tr := trapezoid(f1, x = 1..5, 10)$

> $evalf(tr)$

int(f1, x = 0..1, y = 0..1)

#csak közelítő integrállal lehet kiszámolni

#Alapcsomag: trapéz és érintőformula
#trapezoid(fgv neve, tartomány, részek száma)

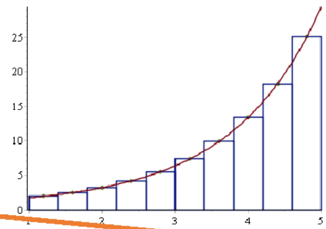
> #simpson formula

#simpson formula
sm := simpson(f1, x = 1..5, 10)

> #a student csomagnak van egy Calculus nevű alcsomagja, amiben van egy ApproximaInt() fgv.

> **Student:-Calculus1:-ApproximateInt(f1, x = 1..5, method = simpson, partition = 10, output = plot)**

Student:-Calculus1:-ApproximateInt(f1, x = 1..5, method = midpoint, partition = 10, output = plot)
#Legpontosabb

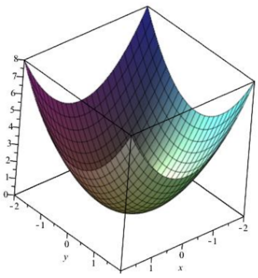


Plot3D:

> *#többváltozós fgv:*
$f1 := x^2 + y^2$ *#forgásparaboloid*

>

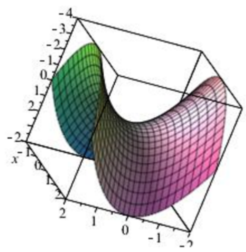
> *plot3d(f1, x = -2..2, y = -2..2)*



with(plots)

#nyereg
$f2 := x^2 - y^2$

plot3d(f2, x = -2..2, y = -2..2)



Animáció:

with(plots)

> $animate\left(\sin\left(x + \frac{p}{15}\right), x = 0..Pi, p = 0..30, frames = 200\right)$

p=0..”elmozdulás maximuma”

> *#f(x)+a függőleges, f(x+a) vizszintes*

> $Interactive\left(sqrt(x^2 + y^2) \cdot \sin\left(\frac{1}{10}\right) \cdot i\right)$

konzol input:

```
#input bekérése terminálról
#lnko és lnkt gyártása

#lnko eljárás, 2 egész számot vár párbeszédpanelből és visszaadja az lnko-t és az lnkt-t.
 #Ha nem egész számot kap, hibakezelés


lnko := proc( )
 local n, m, o, t;

 n := readstat("Add meg az első egész számot");

 while not(type(n, posint))do
  n := readstat("pozitív egész legyen!!");
 od;

 m := readstat("Add meg a második egész számot");
 while not(type(m, posint))do
  m := readstat("pozitív egész legyen!!");
 od;

 o := igcd(n, m);
 t := ilcm(n, m);
 print("legnagyobb közös osztó:", o);
print("legnagyobb közös többszörös:", t);

 end proc
```

$$lnko(\ )$$