Aspecte de programare cu PostScript (eseu)

Vlad Bazon*

iunie 2019

Nu avem aici, un „*alt* tutorial" – că nu plecăm de la Hello, world!, sau „iată un *pătrat*" (ca pentru cazul când n-ai avut de-a face cu nici un limbaj de programare). Prezentăm doar o experienţă proprie (şi pe cât este posibil - originală) de investigare şi înţelegere a limbajului PS (*PostScript*), considerând de la bun început că matematica elementară (şi deprinderea de a pune şi analiza „probleme") este inevitabilă şi asumând (iarăşi, fără cine ştie ce griji didactice) deprinderea de a folosi diverse programe utilitare şi interpretoare (uzuale pe un sistem Ubuntu-Linux), având în vedere că PS este cumva „intim" legat de ecosistemul TeX.

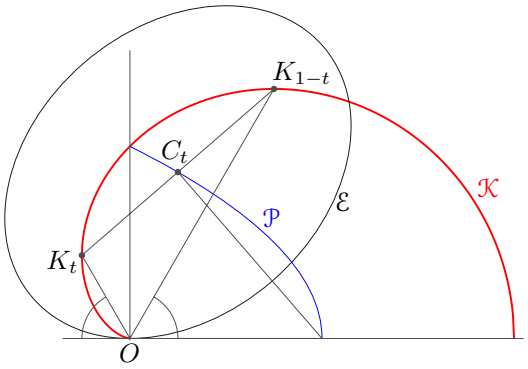
Avem mai degrabă un *eseu*; elementele de limbaj PS nu sunt expuse în mod sistematic şi cu exemplificări simple (ca în tutoriale), ci sunt investigate pe măsură ce ele trebuie practicate pentru a avansa în realizarea proiectului propus iniţial: a constitui o anumită figură geometrică (dar mai complicată decât un pătrat) şi a o eticheta astfel încât să poată fi integrată onorabil (plecând de la un fişier LaTeX) într-un document PDF.

*vlad.bazon@gmail.com

Cuprins

1 Componentele programului (sau proiectului)

Elipsa $\mathcal{E}$ este tangentă axei semicardioidei $\mathcal{K}$ în nodul acesteia O, are focarele K_t şi K_{1-t} situate pe $\mathcal{K}$, iar centrul ei C_t este situat pe arcul de parabolă $\mathcal{P}$ (unde $t \in [0,1]$ generează într-un anumit mod punctele lui $\mathcal{K}$):



Vom mai preciza pe parcurs, contextul matematic al figurii (expus şi separat, în Anexa B). Programul *PostScript* care a produs figura postată mai sus este compus astfel:

- fişierul principal '`grafic.eps`', prin care se generează şi se trasează curbele (elipsa, cardioida şi parabola), segmentele şi arcele importante; vom viza două moduri de generare: „prin puncte" (cu instrucţiuni `lineto`) respectiv, prin cubice Bézier (folosind `curveto`);
- un anumit număr de fişiere EPS asociate etichetelor matematice pe care vrem să le plasăm pe grafic - fişiere pe care le vom obţine „automat", printr-un script *Bash* care iterează pe etichetele respective un anumit şablon de fişier LaTeX, invocând compilatorul **`latex`** şi programul **`dvips`**;
- fişierul '`pack.eps`' care încapsulează în final, fişierele EPS menţionate mai sus; pasându-l programului **`ps2pdf`** rezultă fişierul PDF, precum cel vizualizat grafic mai sus.

Urmează, faţă de acest plan iniţial de lucru, să detaliem, să investigăm şi desigur… să divagăm.

Limbajul *PostScript* (PS) serveşte pentru a descrie o pagină; de regulă, imprimanta va ejecta pagina rezultată prin interpretarea fişierului care i-a fost transmis. EPS rezolvă problema încapsulării de fişiere PS, permiţând includerea în siguranţă (fără ejectare şi fără conflicte între parametrii grafici) a unor pagini PS într-o alta. Pentru a respecta formatul EPS, mai întâi declarăm la începutul fişierelor noastre:

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 75 80 320 248
```

Vom arăta la momentul potrivit, cum obţinem valorile necesare pentru `BoundingBox` (nu neapărat, cele precizate mai sus) şi cum le folosim în vederea încapsulării finale a fişierelor respective.

2 Discuţie despre operatori, variabile şi transformări (pe seama producerii elipsei dintr-un cerc)

Definim întâi o procedură pentru a obţine $\mathcal{E}$, o elipsă tangentă în O axei Ox, având centrul C_t şi semiaxa mică b date, cu semiaxa mare fixată $a = 1$ şi astfel încât panta axei mari este tg $\sqrt{b/a}$:

```
/ellipse {   % tangentă în O axei Ox, cu semiaxele şi centrul date
    /Sb exch def   % Preia din stivă semiaxa mică, b. Semiaxa mare este a=1.
    /yC exch def   % Coordonatele centrului (preluate pe rând,
    /xC exch def   % din stiva operanzilor).
    /savematrix matrix currentmatrix def % salvează CTM
    xC yC translate   % translatează în centrul elipsei
    Sb sqrt 1 atan rotate   % roteşte cu arctg √(b/a)
    1 Sb scale   % scalează orizontal cu a=1 şi vertical cu b
    0 0 1 0 360 arc   % centru (0,0), „rază" 1 (=a pe Ox, =b pe Oy)
    savematrix setmatrix   % restaurează CTM („current transform matrix")
} bind def
```

În limbajele uzuale definiţia de mai sus ar arăta cam aşa: `ellipse(xC, yC, Sb) { ... }`; un apel ca `ellipse(5, 7, 0.75)` creează în memorie un „cadru-stivă" în care se depun valorile 5, 7 şi respectiv 0.75 şi apoi pune în execuţie blocul de instrucţiuni identificat prin '`ellipse`'; în interiorul acestuia, variabilele locale '`xC`', etc. vor fi asignate cu valorile transmise la apel în cadrul-stivă; la încheierea executării codului respectiv, cadrul-stivă asociat la apel va fi eliminat şi controlul va reveni programului apelant.

Spre deosebire de limbajele obişnuite—în care „cadrele-stivă" sunt asociate în mod implicit şi sunt „invizibile"—în *PostScript* stiva are o natură explicită şi sunt prevăzute şi menţinute pe parcursul execuţiei mai multe stive, între care: o stivă pentru operanzi, una pentru operatori sau proceduri şi o stivă (sau „dicţionar") de fonturi.

Definiţia de mai sus este depusă întâi, termen după termen, pe stiva operanzilor: `/ellipse`, `{`, ..., `}` şi când se va întâlni apoi `def`, interpretorul va proceda astfel: va accesa dicţionarul operatorilor şi va căuta aici termenul '`ellipse`'; dacă acesta nu există, atunci va crea o nouă intrare de forma „cheie $\mapsto$ valoare", punând drept „cheie" numele `ellipse` şi drept „valoare" secvenţa de instrucţiuni cuprinsă între '`{`' şi '`}`'.
De aici încolo, când va fi întâlnit ulterior în programul curent, termenul '`ellipse`' va fi interpretat ca „pointer" la secvenţa de instrucţiuni asociată astfel în dicţionarul operatorilor (conducând la executarea acestor instrucţiuni);

dar aceasta *nu* scuteşte etapa prealabilă, de căutare a termenului întâlnit în dicţionarul operatorilor – cu această excepţie: dacă înainte de `def` este întâlnit **`bind`** (ca şi în cazul de faţă), atunci operatorii PS *predefiniţi* întâlniţi în spate sunt înlocuiţi imediat prin „pointeri" la codul asociat lor ca „valoare" în dicţionarul operatorilor (scutind în acest caz, căutarea ulterioară în dicţionar; se accelerează astfel, execuţia procedurii în care sunt invocaţi).

Dacă după definiţia de mai sus ar urma o invocare ca de exemplu `5 7 0.75 ellipse`, interpretorul ar depista cheia '`ellipse`' în dicţionarul operatorilor şi ar începe să execute instrucţiunile care i-au fost asociate; primele trei instrucţiuni, `/Sb exch def` etc., sunt interpretate conform mecanismului general descris mai sus, dar cu o anumită modificare (de observat că acum lipsesc acoladele care încadrează de obicei blocul de instrucţiuni al procedurii): se înfiinţează în dicţionarul operatorilor câte o nouă cheie ('`Sb`' etc.), căreia i se asociază ca „valoare" *nu* pointerul la codul operatorului predefinit `exch` (ca în cazul când s-ar fi întâlnit `{exch}`), ci efectul executării acestuia urmată de descărcarea din stiva operanzilor determinată de operatorul final `def`. Sunt modelate astfel atribuirile uzuale, `Sb = 0.75`, etc.

Ne putem clarifica mecanismele folosite în *PostScript* experimentând în sesiuni de lucru cu Ghostscript. De exemplu, să introducem pe stivă coordonatele unui punct, (5, 7) şi să imităm mecanismul de „atribuire" tocmai descris pentru a crea o variabilă `y` legată la ordonata punctului (modelând atribuirea uzuală y=7):

```
vb@Home:~/keps$ gs -q
GS> 5 7 /y
GS<3> pstack   % afişează conţinutul stivei, începând din vârful acesteia
/y
7
5
GS<3> exch pstack   % 'exch' schimbă între ele primele două valori din stivă
7
/y
5
GS<3> def pstack   % 'def' mută în stiva operatorilor cheia 'y', asociindu-i valoarea 7
5   % în stivă a rămas doar abscisa punctului
GS<1> y =      % operatorii '=' şi '==' afişează valoarea asociată cheii indicate
7
GS<1> quit
```

În procedura `ellipse` redată mai sus, elipsa este descrisă pe baza operatorului predefinit `arc`; acesta are sintaxa „$x_c\ y_c\ r\ \alpha_1\ \alpha_2$ **arc**" şi are ca efect vizual producerea unui arc de cerc de rază r, cu centrul (x_c, y_c), începând în sens antiorar de la α_1° şi până la α_2°; pentru $\alpha_1 = 0$ şi $\alpha_2 = 360$ se obţine întregul cerc („efectul" real este crearea unei variabile de memorie—de tipul '`path`', sau „contur"—conţinând coordonatele punctelor pe baza cărora se vor trasa curbele Bézier necesare; trasarea efectivă se declanşează prin invocarea ulterioară a operatorului **`stroke`**).

Dar cât de „mare" să fie cercul constituit astfel? Raza `r` este un număr

(abstract, în matematică) şi n-avem decât să-l mărim dacă vrem un cerc „mai mare”; dar în tipografie, când avem de-a face cu o pagină de hârtie (sau cu o pagină de ecran), unitatea de măsură devine esenţială. În *PostScript* pagina de hârtie este asociată cadranului întâi al unui sistem de coordonate xOy cu axa Ox peste marginea de jos a paginii şi axa Oy peste marginea stângă a acesteia; *unitatea de măsură* este $1\,\mathrm{bp} = \frac{1}{72}\,\mathrm{inch}$ (1 *inch* = 2.54 centimetri); arcele sunt măsurate în grade.

Sunt prevăzute *transformări ale sistemului de coordonate*, în două variante (aparent, distincte): putem utiliza matrici (de legătură între coordonatele vechi şi cele noi), având astfel şi un procedeu simplu pentru a salva şi a restaura sistemul de coordonate curent (salvăm „matricea de transformare curentă”); sau, putem folosi operatorii specializaţi `translate` (care mută originea în punctul indicat), `rotate` (care roteşte axele în jurul originii actuale, cu unghiul indicat) şi `scale` (prin care se poate stabili o nouă unitate de măsură, pe o axă, pe cealaltă, sau pe ambele axe de coordonate).

Operatorul `arc` produce un cerc (sau un arc de cerc) în sistemul de coordonate *iniţial*, dar dacă scalăm în prealabil cu a pe orizontală şi cu b pe verticală atunci, în sistemul de coordonate rezultat va fi produsă o elipsă cu semiaxele a şi b.

În cazul nostru, a fost necesar să rotim elipsa în jurul centrului său, cu un anumit unghi şi a trebuit să ţinem seama de faptul că `rotate` roteşte axele în jurul *originii* sistemului de coordonate curent (nu în jurul unui alt punct) - ordonând transformările necesare astfel: întâi am translatat originea iniţială în punctul ale cărui coordonate le preluasem de pe stivă în `xC` şi `yC`; apoi am rotit în jurul noii origini cu unghiul arctan $\sqrt{b}$; apoi am folosit `1 Sb scale`, lăsând unitatea de măsură pe orizontală egală cu 1 pentru că semiaxa mare este 1 şi punând-o egală cu `Sb` (semiaxa mică) pe verticală; în final, am invocat `0 0 1 0 360 arc` - obţinând „cerc” cu centrul (0,0) (adică originea sistemului de coordonate *curent*) şi de „rază” 1 (dar 1 pe verticală înseamnă acum valoarea din `Sb`, deci rezultă nu cerc, ci elipsă).

De încă un lucru a trebuit să mai ţinem seama: cu ce *grosime* urmează să trasăm elipsa obţinută? Grosimea implicită a liniei este de 1`bp`[1] şi poate fi setată altfel prin operatorul `setlinewidth`; dar valoarea respectivă se raportează la scalarea *curentă*, astfel că în cazul nostru (când unitatea de măsură diferă între cele două axe) elipsa ar fi trasată (când se va invoca `stroke`) cu o „peniţă” de grosime variabilă. Pentru a evita aceasta, înainte de a efectua transformările arătate mai sus am salvat *matricea transformării curente* (desemnată prin CTM) şi am restaurat-o în final, după producerea elipsei; procedând astfel, scalările întreprinse în interior sunt uitate.

Pentru a testa procedura `ellipse`, adăugăm în fişierul '`grafic.eps`' de exemplu:

```
72 72 scale   % noua unitate de măsură este de 72bp (= 1 inch = 2.54cm)
```

[1] într-adevăr, `GS> currentlinewidth pstack` ne dă `1.0`)

```
2 6 2 ellipse   % centrul este la 2in de marginea stângă şi 6in de baza paginii
.3 setlinewidth   % grosimea „reală” a liniei este 0.3*72=21.6bp
.4 setgray   % nuanţa de negru pentru trasarea liniei
stroke   % trasează conturul (pe ecran, sau pe hârtie)
```

Transmiţând fişierul imprimantei—`lpr grafic.eps`—obţinem o pagină conţinând elipsa respectivă[2]. Am prevăzut o grosime aşa de mare (21.6bp = 0.762 milimetri) pentru trasarea elipsei, în scopul de a testa cum arată elipsa dacă n-am mai salva/restaura CTM:

În stânga imaginii avem elipsa produsă prin programul de mai sus (cu salvare şi restaurare CTM), iar în dreapta aceeaşi elipsă (deplasată puţin spre dreapta, prin `translate`), produsă printr-o dublură a programului în care însă, am eliminat cele două linii pentru salvare/restaurare CTM. Desigur, aici am redat imaginea cu o anumită micşorare (pe hârtie, grosimea liniei pentru prima elipsă *este* de 7-8 milimetri), dar se vede suficient de bine că în cazul elipsei din dreapta grosimea liniei este variabilă (ajungând de două ori mai mare pe axa mare, decât pe axa mică).

3 Producerea şi trasarea liniilor (fişierul `grafic.eps`)

În §2 am descris elipsa asumând că semiaxa mică şi centrul ei sunt *transmise* prin stivă (vrând să experimentăm şi să lămurim aspecte privitoare la stive, proceduri, variabile şi transformări PS). De fapt, toate elementele necesare programului '`grafic.eps`' planificat în §1, decurg din descrierea parametrică a punctelor semicardioidei $\mathcal{K}$ (v. Anexa B):

$$K(t) = \left(X(t) = 2t(2t-1),\, Y(t) = 4t\sqrt{t(1-t)} \right),\ t \in [0,1]$$

Semiaxa mare a lui $\mathcal{E}$ este $a = 1$, semiaxa mică este $b = 4t(1-t)$ iar focarele sunt punctele $K(t)$ şi $K(1-t)$. Centrele elipselor $\mathcal{E}$ (când t variază) aparţin arcului de parabolă $\mathcal{P}$ ale cărui puncte sunt date de ecuaţia $y = \sqrt{1-x}$, pentru $x \in [0,1]$.

Corespunzător, în `grafic.eps` am avea aceste definiţii iniţiale:

[2] precizarea „dacă imprimanta dispune de un interpretor propriu de PS” a devenit inutilă, în zilele noastre

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 50 53 387 165

% K(t) are ecuaţiile: X(t) = 2t(2t − 1), Y(t) = 4t√(t(1 − t)), t ∈ [0, 1]
/X {2 mul dup 1 sub mul} bind def  % t este dat în vârful stivei
/Y {dup dup 1 exch sub mul sqrt mul 4 mul} bind def

% focarele K(t) şi K(1-t), semiaxa mică, centrul elipsei şi arcele razelor focale
/tL 0.25 def  % un t între 0 şi 1 (vizând arcul lui K(t) din stânga axei Oy)
/tR {1 tL sub} bind def  % 1-t (vizând arcul lui K(t) din dreapta axei Oy)
/Sb {4 tL tR mul mul} bind def  % semiaxa mică a elipsei, Sb = 4t(1-t)
/x1 {tL X} def   /y1 {tL Y} def  % coordonatele punctului K(tL)
/x2 {tR X} def   /y2 {tR Y} def  % coordonatele punctului K(tR)
/xC {x1 x2 add 2 div} bind def  % coordonatele mijlocului C(t) al
/yC {y1 y2 add 2 div} bind def  % segmentului K(tL)—K(tR)
/arc2 {y2 x2 atan} bind def  % arcul dintre Ox şi raza focarului K(tR)

% vom marca unele puncte, prin mici discuri
/rp {1.2 72 div} bind def  % raza discului prin care marcăm un punct
/punct {0.3 setgray  rp 0 360 arc fill} bind def
```

Am ales pentru t o valoare convenabilă, `tL=0.25`; am obţinut coordonatele punctelor `K(tL)` şi `K(tR=1-tL)` (aflate pe $\mathcal{K}$ în stânga şi în dreapta axei Oy), aplicând operatorii `X` şi `Y` valorilor `tL` şi `tR`.

`y2 x2 atan` dă $\alpha_2^\circ = \operatorname{arctg}\frac{y2}{x2}$, astfel că `0 0 0.25 0 arc2 arc` va produce arcul de cerc cu centrul în originea sistemului curent de coordonate, cu raza 0.25 (faţă de unitatea de măsură aflată în uz în momentul trasării ulterioare, prin `stroke`), începând în sens antiorar de la axa Ox şi încheiat la α_2°. Arcele produse astfel prin `arc2` şi `arc1`=180°`-arc2` vor marca unghiurile cu Ox ale razelor polare corespunzătoare focarelor lui $\mathcal{E}$ (unghiuri care sunt *egale*, însemnând că $\mathcal{E}$ este *tangentă* în O la Ox).

Pentru a putea marca anumite puncte, am definit procedura `punct` (nu există un operator *predefinit* care să „traseze" un punct): folosind `arc fill`, se trasează un disc (cerc umplut cu o nuanţă de gri închis) cu raza foarte mică (puţin peste `1bp`), având centrul în punctul ale cărui coordonate vor fi preluate de pe stivă.

Procedurile introduse mai sus `xC`, `yC` şi `Sb` ne dau, pentru $\mathcal{E}$, coordonatele centrului şi semiaxa mică - încât putem acum înlocui procedura '`ellipse`' din §2, cu:

```
% elipsa cu semiaxele a=1 şi b=Sb, centrată în (xC,yC), tangentă în O la Ox
/Ellipse {
    /savematrix matrix currentmatrix def % salvează CTM
    xC yC translate  % translatează în centrul elipsei
    Sb sqrt 1 atan rotate  % roteşte cu arctan √(b/a)
    1 Sb scale  % scalează orizontal cu a şi vertical cu b
    0 0 1 0 360 arc  % „cerc" de centru (0,0), „rază" 1 (=a pe Ox şi =b pe Oy)
    savematrix setmatrix  % restaurează CTM („current transform matrix")
} bind def
```

Producem semicardioida prin puncte cât mai apropiate (unite prin segmente), generând (prin **repeat**) N=1000 de puncte ale lui $\mathcal{K}$:

```
% aproximare poligonală (cu N=1000 de segmente) a semicardioidei
/Kardioid {
    /N 1000 def
    /dt 1 N div def   % pasul dt = 0.001
    /t 0 def          % t va parcurge intervalul (0,1), cu pasul dt
    0 0 moveto
    N { /t t dt add def   % t = t+dt
        t X t Y lineto    % segment de la K(t) la K(t+dt)
    } repeat
} bind def
```

Generăm (tot „punct cu punct") şi $\mathcal{P}$, unind punctul curent (iniţializat în vârful parabolei) cu cel corespunzător prin $\sqrt{1-x}$ abscisei depuse pe stivă la iteraţia următoare (prin mecanismul asigurat de `for`), coborând abscisa cu câte 0.01 dinspre 1 spre 0 (obţinând o aproximare cu 100 de segmente a lui $\mathcal{P}$):

```
% arcul de parabolă y = sqrt(a(a-x)), x ∈ [0,a] unde aici, a = 1
/Parabola {
    1 0 moveto   % vârful parabolei este punctul (1,0)
    1 -.01 0 {
        dup neg 1 add sqrt lineto
    } for
} bind def
```

Precizăm că `From Step To {Proc} for` decurge cam aşa: se depune pe stivă valoarea iniţială `From` (aici, abscisa 1) şi se execută `Proc`; apoi valoarea din vârful stivei este „majorată" cu `Step` (aici cu `-0.01`, rezultând abscisa utilizată în următoarea iteraţie în `Proc`) şi dacă astfel nu se „depăşeşte" `To`, se repetă executarea secvenţei `Proc`; ş.a.m.d. Pe parcurs, în vârful stivei găsim abscisa x curentă (1, apoi 0.99, 0.98, ..., 0), pe care `Proc` o poate folosi adăugând-o încă o dată în stivă (prin **dup**).

Bineînţeles că putem testa `grafic.eps` în orice moment, adăugând o secvenţă în care mărim convenabil unitatea de măsură, translatăm mai spre centrul paginii, setăm o anumită grosime de trasare şi apoi invocăm procedurile pentru $\mathcal{E}$, $\mathcal{K}$ şi $\mathcal{P}$:

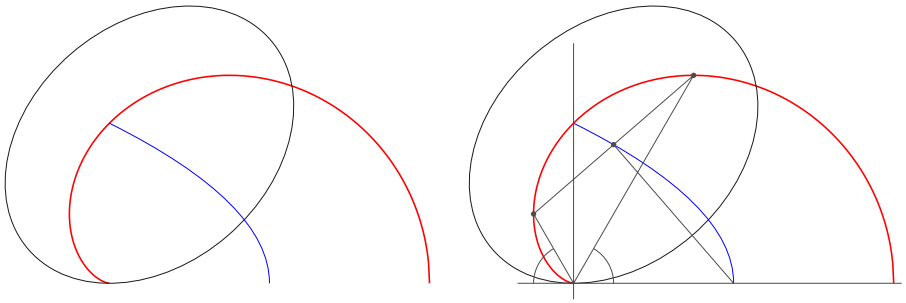
```
% „programul principal": scalează, poziţionează, trasează liniile
/um 60 def   % unitatea de măsură
um um scale
1.5 1 translate   % originea figurii: 1.5um de marginea stângă, 1um de jos

.01 setlinewidth 1 0 0 setrgbcolor Kardioid stroke
.005 setlinewidth  0.1 setgray Ellipse stroke
0 0 1 setrgbcolor  .005 setlinewidth Parabola stroke
```

Obţinem prima figură dintre cele două redate mai jos; ne-a rămas de adăugat focarele şi centrul elipsei, segmentele aferente acestora şi cele două arce egale prin care sugerăm că $\mathcal{E}$ este tangentă axei orizontale; ca să evităm

„apelarea” separată, constituim o procedură ’`other_lines`’ care să conţină toate aceste construcţii:

```
/other_lines {
    % marchează individual punctele K(t), K(1-t) şi C(t)
        x1 y1 punct
        x2 y2 punct
        xC yC punct
    % triunghiul K(tL) — K(tR) — origine (— closepath)
        x1 y1 moveto   % K(t)
        x2 y2 lineto   % K(1-t)
        0 0 lineto closepath
    % axele Ox, Oy; uneşte vârful parabolei cu centrul elipsei
        2.05 0 −0.35 0 moveto lineto   % Ox (de la abscisa -0.3 până la 2.05)
        0 1.5 0 −0.1 moveto lineto   % Oy (vertical -0.1 .. 1.5)
        1 0 xC yC   moveto lineto   % segmentul C(t)—(1,0)
    % arcele dintre razele focarelor şi Ox
        0 0 0.25 0 arc2   arc stroke
        0 0 0.25 180 arc2 sub 180   arc stroke
} bind def
```

Translatând sistemul de coordonate spre dreapta (de exemplu prin `2.9 0 translate`), repetând cele trei linii din programul principal prin care se trasează $\mathcal{K}$, $\mathcal{E}$ şi $\mathcal{P}$ şi adăugând linia `0 setlinewidth other_lines stroke`, obţinem şi a doua figură:



Rămâne să *etichetăm* focarele, centrul elipsei şi eventual, curbele respective; desigur, fontul angajat pentru etichete ar trebui să se potrivească (ca tip de literă şi ca mărime) celui folosit în text.

Dar înainte de aceasta, să precizăm că figura inserată mai sus este un fişier PDF (şi nu o copie-ecran—fişier PNG—care arăta mai rău, pe lângă faptul că ar fi fost de câteva ori mai voluminos), obţinut într-o manieră tipică:

```
vb@Home:~/keps$ gs −q −o /dev/null −sDEVICE=bbox grafic.eps
    %%BoundingBox: 50 53 387 165

vb@Home:~/keps$ ps2pdf grafic.eps   # rezultă 'grafic.pdf'

vb@Home:~/keps$ gs −sDEVICE=pdfwrite −o keps1.pdf −f grafic.pdf \
                  −c "[/CropBox [50 53 387 165] /PAGES pdfmark"
```

Am invocat **gs** cu **bbox** pentru a obţine valoarea %%BoundingBox pentru figura creată prin programul 'grafic.eps' şi apoi cu **pdfwrite** pentru a insera (prin mecanismul '**pdfmark**') declaraţia /CropBox (corespunzătoare cu %%BoundingBox) în fişierul PDF rezultat prin **ps2pdf**.

3.1 Eroare, experiment şi descoperire

Situaţiile de „eroare", în pofida aparenţei de „surpriză neplăcută", sunt instructive; provocând chiar, asemenea situaţii şi experimentând în contextul respectiv, poţi ajunge să clarifici aspecte care de obicei sunt vizate expeditiv, ici şi colo.

Dacă în procedura Kardioid din 'grafic.eps' înlocuim „/N 1000 def" de exemplu, cu „/N 10 def", atunci obţinem un mesaj de eroare:

```
vb@Home:~/keps$ gs -q  grafic.eps
Error: /rangecheck in --sqrt--
Operand stack:
   2.0    1.0    -1.19209e-07
Execution stack:    ...
```

Chiar neplăcut... pentru 1000 merge, dar pentru *10* nu ?! Dar să profităm de faptul că N este mic (şi să ignorăm deocamdată, mesajul afişat de *GS*): inserăm nişte instrucţiuni de afişare intermediară a unor valori curente, folosind operatorul ' == ' (atenţie: spaţiul iniţial şi cel final sunt obligatorii!). În cazul de faţă am insera 't == ' în instrucţiunea repetitivă 'N {...} repeat' (unde acum N=10):

```
N { /t t dt add def   % t = t+dt (unde dt = 1/N = 0.1)
      t ==   % afişează valoarea t curentă
      t X t Y lineto    % segment de la P(t) la P(t+dt)
} repeat
```

şi după aceasta putem vedea cum decurg lucrurile:

```
vb@Home:~/keps$ gs -q  grafic.eps
0.1
0.2
0.3
0.4
0.5
0.6
0.700000048   % 0.6 + 0.1 (!)
0.800000072
0.900000095
1.00000012
Error: /rangecheck in --sqrt--
...
```

Procedura eşuează imediat după ce t ajunge la valoarea 1.00000012, deci la calculul coordonatelor X(t) şi Y(t) ale capătului segmentului curent; ajunge să te uiţi la definiţiile iniţiale, pentru a vedea despre ce este vorba: Y(t) = 4t sqrt(t(1-t)), ori pentru ultima valoare a lui t avem 1-t < 0

– ceea ce determină semnalarea de la începutul mesajului de eroare, pe care acum o putem şi simula direct:

```
vb@Home:~/keps$ gs -q
GS> 1 1.00000012 sub pstack sqrt
-1.1920929e-07  % rezultatul scăderii 1 - 1.00000012
Error: /rangecheck in --sqrt--
Operand stack:
    -1.19209e-07
...
```

Dar altceva este interesant de observat, în experimentul de mai sus: `t` avea valoarea iniţială 0 şi este incrementat la fiecare iteraţie cu 0.1; însă după `0.6` urmează mai sus `0.700000048` şi nu 0.7! Propagarea acestor erori de aproximare *explică* faptul că se depăşeşte limita 1, ajungându-se în situaţia de eroare evidenţiată mai sus.

Majoritatea limbajelor de programare (inclusiv *PostScript*) mizează pe o reprezentare floating point a numerelor reale (o excepţie notabilă este limbajul TeX, creat pe când încă nu apăruse standardul „floating point": pentru a repera cât mai fin „punctele" unei pagini de hârtie, Knuth a introdus reprezentarea numerelor reale ca multipli întregi de 2^{-16}).

Din cauza erorilor de aproximare inerente, se recomandă ca variabila care asigură iterarea în cadrul unei secvenţe repetitive să fie de tip *întreg*.

Şi în *PostScript* avem de distins între reprezentarea (şi operarea) internă a numerelor şi pe de altă parte, afişarea acestora - cum arată acest mic experiment:

```
vb@Home:~/keps$ gs -q
GS> /h 0.1 def  % h = 0.1
GS> /g {h 0.6 add} def  % g = h + 0.6
GS> g =
0.7  % valoarea afişată pentru liniştea noastră...
GS> g ==
0.700000048  % valoarea utilizată intern
```

' `=` ' afişează dacă se poate, cam cum ne-am dori; în schimb ' `==` ' (şi încă, ' `===` ' pentru anumite alte obiecte PS) produce pe ecran valoarea *internă* existentă (şi aceasta este cea utilizată în calcule).

3.2 De la semicardioidă la cardioidă, cu `pathforall`

`Kardioid` instituie în memorie conturul poligonal al *semicardioidei* superioare; vom arăta că prin `pathforall` putem obţine de aici, conturul întregii cardioide.

Să constituim un fişier nou, '`cardioid.eps`', în care să rescriem deocamdată procedura `Kardioid`, având grijă de această dată să ghidăm iteraţiile prin întregi:

```
%!PS
% coordonatele punctelor semicardioidei (t fiind în vârful stivei)
/X {2 mul dup 1 sub mul} bind def   % X(t) = 2t(2t - 1)
/Y {dup dup 1 exch sub mul sqrt mul 4 mul}
    bind def   % Y(t) = 4t sqrt(t(1 - t))
/N 1000 def
% aproximare poligonală (cu N segmente) a semicardioidei superioare
/Kardioid {
    0 0 moveto   % conturul pleacă din t=0 (vârful cardioidei)
    1 1 N { N div dup X exch Y lineto } for
} bind def
```

`1 1 N {N div dup X exch Y lineto} for` funcţionează cam aşa: la fiecare iteraţie se depune pe stivă valoarea curentă a indexului, `i` = 1..N; `N div` adaugă `N` pe stivă, împarte `i` la `N`, elimină cei doi operanzi din stivă şi pune în loc rezultatul `i/N`; mai departe, **`dup`** adaugă a doua oară în stivă `i/N`, iar `X` calculează abscisa punctului K(`i/N`) şi pune rezultatul în locul acestei copii - încât acum stiva conţine vechea valoare `i/N` şi deasupra acesteia, abscisa rezultată; `exch` interschimbă între ele aceste două valori, încât apoi, `Y` poate calcula ordonata lui K(`i/N`); după aceasta, în stivă avem abscisa şi ordonata punctului K(`i/N`), iar `lineto` va adăuga conturului segmentul de la ultimul său punct, la K(`i/N`).

„Conturul" final este format din instrucţiunea '`0 0 moveto`' şi N=1000 de instrucţiuni `lineto` pentru segmentele care unesc K(`(i-1)/N`) cu K(`i/N`), unde `i=1..N`. Putem „vedea" conturul (adică, instrucţiunile constituente), folosind operatorul **`pathforall`**; trebuie să-i transmitem pe stivă *patru proceduri*, vizând respectiv instrucţiunile `moveto`, `lineto`, `curveto` şi `closepath` din componenţa conturului curent; în fiecare caz, sunt plasate pe stivă coordonatele punctelor angajate în instrucţiune şi apoi se execută procedura transmisă pentru acel tip de instrucţiune.

De exemplu, dacă după ce am constituit un contur, am invoca `{ = = } {} {} {} pathforall`, atunci s-ar afişa coordonatele din toate instrucţiunile `moveto` ale conturului respectiv (şi numai acestea, fiindcă celelalte trei proceduri transmise sunt vide): pentru fiecare instrucţiune `x y moveto` existentă în conturul curent, `pathforall` va pune în stivă `x y` şi va executa procedura indicată `{ = = }`, ceea ce extrage pe rând cele două valori din stivă şi le afişează *una sub alta* (' `=` ' este suficient pentru aceasta, dar mai bine este să folosim ' `==` ' – cum am arătat la §3.1).

Afişarea „una sub alta" a valorilor este foarte convenabilă dacă intenţionăm să le preluăm ulterior într-un alt program (indiferent în ce limbaj), pentru anumite prelucrări independente; dar putem să le afişăm şi într-un format mai „convenabil", înlocuind '`{ = = }`' cu `{[3 1 roll] == }`: se adaugă în stivă obiectul (de tip „*array*") `[...]`, încât acum stiva conţine `x y [3 1 roll]`; apoi se execută instrucţiunea din interior `3 1 roll`, prin care conţinutul stivei este *rotit* spre dreapta şi este înlocuit cu un singur „*array*", anume `[x y]` (pentru a cărui afişare este acum *necesar* ' `==` '). Putem verifica

direct, în *GS*: de exemplu, GS> `7.2 5.6 [3 1 roll] ==` produce GS<1> `[7.2 5.6]`.

Însă `pathforall` poate fi utilizat şi în scopuri mai interesante decât simpla afişare de valori – anume, pentru modificarea sau extinderea conturului curent. Să adăugăm în fişierul `cardioid.eps` o secvenţă prin care prevedem o unitate de măsură şi o grosime de linie convenabile, mutăm originea undeva mai la mijlocul paginii şi apoi, invocăm procedura `Kardioid`, obţinând conturul semicardioidei superioare; invocând `pathforall` cu o procedură pentru instrucţiunile `x y lineto` în care schimbăm semnul ordonatelor (folosind `neg`), conturului curent îi vor fi adăugate noile instrucţiuni `x -y lineto` încât în final, conturul va conţine segmentele iniţiale ale semicardioidei superioare, dar şi pe cele simetrice acestora faţă de Ox. În final, afişăm conturul astfel „întregit”, folosind din nou `pathforall`:

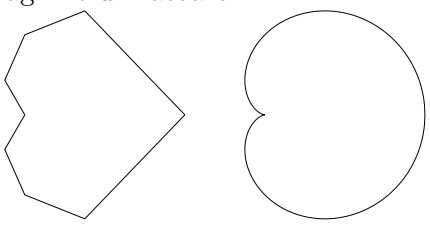
```
90 90 scale   % 90bp = 72bp + 18bp = 1.25 inch (= 4.175 cm)
1.5 5 translate   % originea figurii: 135bp de marginea stângă, 450bp de jos
.005 setlinewidth   % 0.005 din 90bp

Kardioid   % constituie conturul poligonal al semicardioidei superioare

% adaugă „aceleaşi” segmente, dar schimbând semnul ordonatelor
{moveto} {[3 1 roll neg lineto]} {} {}
    pathforall   % rezultă conturul întregii cardioide

% afişează instrucţiunile conturului curent (după „întregire”)
{[3 1 roll] == }   % [x y] din instrucţiunile 'moveto' ale conturului
{[3 1 roll (lineto)] == }   % [x y lineto]
{} {}
    pathforall

clear   % curăţă stiva (cum-necum, au rămas 4 tablouri vide...)
stroke   % trasează efectiv cardioida (nu numai semicardioida!)
```

Pentru exemplu, să redefinim `/N 4 def` şi să lansăm programul sub *GS*:

```
vb@Home:~/keps$ gs -q  cardioid.eps
[2.53962298e-05 1.61245898e-05]   % moveto
[-0.250021845 0.433032304 (lineto)]
[2.53962298e-05 0.999998689 (lineto)]
[0.750012338 1.29901314 (lineto)]
[1.99999058 1.61245898e-05 (lineto)]
[2.53962298e-05 1.61245898e-05]     % moveto
[-0.250021845 -0.433051646 (lineto)]
[2.53962298e-05 -1.000018 (lineto)]
[0.750012338 -1.29903245 (lineto)]
[1.99999058 -3.54741e-05 (lineto)]
GS>
```

Se vede că au rezultat nu numai cele 4 segmente care „aproximează” semicardioida superioară, dar şi simetricele acestora faţă de Ox; de observat însă, că `neg` nu a schimbat pur şi simplu semnele, ci a şi evaluat cumva, rezultând

obişnuitele „erori de aproximare” (astfel, `0.433032304` a ajuns `-0.433051646` ceea ce diferă de valoarea corectă `-0.433032304`, cu vreo două sutimi de miime). În orice caz, pentru N=4 şi respectiv N=1000 obţinem (micşorând însă unitatea de măsură, la 30*bp*) imaginile următoare:



Subliniem că trasarea conturului pentru N=1000 necesită execuţia a 2002 instrucţiuni (două `moveto` şi 1000 + 1000 instrucţiuni `lineto`); vom arăta acuşi că putem trasa cardioida (şi alte curbe) cam cu aceeaşi acurateţe (totuşi… arătând mai bine!), dar cu un contur conţinând mult mai puţine instrucţiuni (anume, instrucţiuni `curveto`). Am avansa ideea cam aşa: dacă în loc de cele 4 segmente din prima figură am considera nişte arce curbate cumva „din ochi”, sau din ecuaţiile curbei iniţiale, atunci conturul constituit de acestea ar fi oricum „mai bun”, decât cel poligonal iniţial.

3.3 De la semicardioidă la cardioidă, simetrizând faţă de axă

Mai sus am urmat o idee exotică: am extins prin `pathforall` conturul semicardioidei superioare, obţinând conturul întregii cardioide (pentru fiecare instrucţiune `x y lineto` existentă, am adăugat conturului iniţial şi `x -y lineto`); dar astfel, am putut evidenţia ce este de fapt, un „contur”.
Să revenim la calea *firească*: „întregim” cardioida simetrizând faţă de axa Ox conturul semicardioidei (în loc să-l extindem, cum am făcut anterior).

Întâi avem de subliniat că nu merge o schemă de lucru în care conturul să fie instanţiat o singură dată:

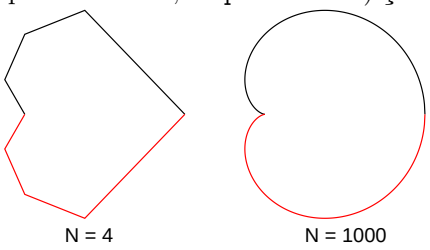
```
Kardioid   % adaugă conturului curent şi semicardioida superioară K
gsave  stroke  grestore   % trasează conturul curent (K se păstrează)
gsave
      1 -1 scale   % inversează sensul pe axa Oy (vrând semicardioida inferioară)
      stroke   % trasează conturul curent; K nu este inversat!
grestore
```

`stroke` trasează conturul *curent* şi trebuie să se ţină seama de situaţia obişnuită, când acesta conţine şi alte contururi, nu numai pe acela căruia, în prealabil, ai vrea să-i aplici o anumită transformare. Decurge firesc această regulă: după ce se indică transformarea, trebuie precizat cumva şi conturul căruia îi va fi aplicată aceasta; perechea de operatori `gsave` şi `grestore` deservesc şi ei, acest principiu de lucru (permiţând comutarea temporară într-un context grafic nou).

Ţinând seama de regula menţionată, programul din §3.2 se rescrie astfel:

```
%!PS
% coordonatele punctelor semicardioidei K(t) (t fiind în vârful stivei)
/X {2 mul dup 1 sub mul} bind def
/Y {dup dup 1 exch sub mul sqrt mul 4 mul} bind def

/N 1000 def  % /N 4 def
% aproximare poligonală (cu N segmente) a semicardioidei superioare
/Kardioid {
    0 0 moveto  % conturul pleacă din K(0) (vârful cardioidei)
    1 1 N { N div dup X exch Y lineto } for
} bind def

90 90 scale  % 90bp = 72bp + 18bp = 1.25 inch (= 4.175 cm)
1.5 5 translate  % originea: 135bp de marginea stângă, 450bp de jos
.005 setlinewidth  % 0.005 din 90bp

gsave
    Kardioid  % instanţază (şi trasează) conturul semicardioidei
    stroke
grestore

gsave
    1 -1 scale  % (x,y) -> (x,-y) (simetrie faţă de axa Ox curentă)
    Kardioid  % instanţiază (pentru noul xOy), conturul semicardioidei
    1 0 0 setrgbcolor  stroke
grestore
```

Iterând programul pentru N=4 (caz în care este uşor de văzut conţinutul conturului înainte şi după simetrizare, cu `pathforall`) şi N=1000 obţinem:



Acum, fiindcă am folosit un *acelaşi* contur de două ori, putem şi distinge (aici, prin culoare) între cele două semicardioide (spre deosebire de §3.2, unde întâi am „întregit" conturul şi apoi l-am plotat ca atare, ceea ce exclude posibilitatea marcării separate a părţilor). Să observăm însă că de executat sunt tot 2N+2 instrucţiuni: câte una `moveto` şi câte N `lineto` pentru fiecare dintre cele două instanţe ale conturului.

4 Etichetarea directă, folosind un font obişnuit

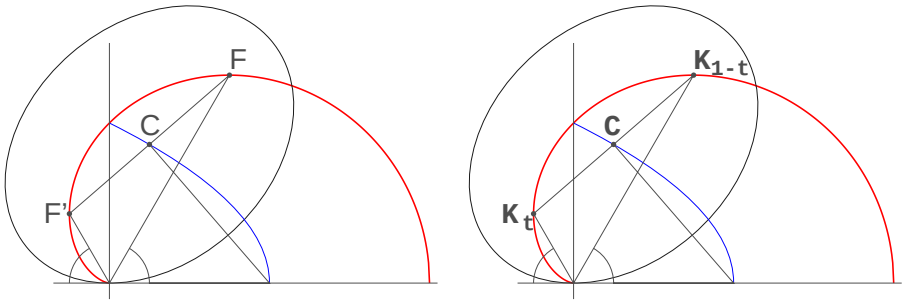
Pentru a eticheta anumite elemente ale graficului poate fi suficient să angajăm un font obişnuit; adăugăm în '`grafic.eps`', de exemplu:

```
/Helvetica findfont 0.18 scalefont setfont
(F) x2 y2 0.04 add moveto show  % eticheta 'F' pentru punctul K(tR)
```

```
(F') x1 0.16 sub y1 0.06 sub moveto show  % dar „prim" va fi ca o virgulă
(C) xC 0.06 sub yC 0.06 add moveto show
```

Am mărit puţin ordonata (cu 0.04 din unitatea de măsură curentă, prevăzută în „programul principal" din `grafic.eps`), am mutat aici „punctul curent" şi apoi **`show`** a afişat caracterul rămas în vârful stivei, `(F)`; am procedat analog, pentru celelalte etichete. Obţinem astfel *prima* dintre cele două figuri din fişierul PDF vizualizat mai jos.

Ar fi de făcut nişte precizări: am indicat numele fontului, '`/Helvetica`' (prefixat cu '`/`', încât să nu fie considerat ca invocare de operator); `findfont` caută în dicţionarul de fonturi cheia indicată şi depune pe stivă *dicţionarul* asociat ei; acesta conţine între altele câte un operator pentru fiecare caracter (denumit ca şi caracterul, 'A', 'B', etc.); acest operator „desenează" caracterul respectiv, folosind ca de obicei când este de trasat un grafic `moveto`, `lineto` şi `curveto`; este asumat sistemul de coordonate curent şi fiindcă unitatea de măsură setată în `grafic.eps` era „mare" ($1 \equiv 60$`bp`), am folosit `scalefont` - obţinând caractere cu o dimensiune „normală", 0.18×60`bp` ≈ 11`bp`. Dicţionarul astfel setat în stiva operanzilor este apoi transferat prin `setfont` în vârful stivei dicţionarelor de fonturi, devenind astfel „fontul curent" pentru viitoarele instrucţiuni „`(`*text*`) show`".



Pe de altă parte, ar fi de dorit ca etichetele să nu fie nişte simple repere, ci să fie astfel formulate încât să reflecte cât mai bine contextul matematic al figurii; de exemplu, notând K_t şi $K_{1\text{-}t}$ în loc de F' şi F – s-ar evidenţia faptul că focarele sunt puncte ale semicardioidei $\mathcal{K}$, deduse dintr-o parametrizare a acesteia cu valori t simetrice faţă de mijlocul 0.5 al intervalului [0, 1].

Pentru acest caz, când ar trebui doar să adăugăm nişte indici, este încă uşor să folosim un font obişnuit (rezultând figura din dreapta):

```
/Courier-Bold findfont 0.18 scalefont setfont
(K) x2 y2 0.04 add moveto show
0 -0.04 rmoveto  % coboară puţin punctul curent, pentru a adăuga indicele t
gsave
.75 .75 scale  % mărimea indicelui va fi 3/4 din mărimea curentă de caracter
(1-t) show
grestore
(K) x1 0.2 sub y1 0.06 sub moveto show
```

```
0.02 -0.04 rmoveto
gsave
.75 .75 scale
(t) show
grestore
(C) xC 0.06 sub yC 0.06 add moveto show
```

Desigur, am ambalat modificarea mărimii curente de caracter (rezultată aici prin scalarea sistemului de coordonate) şi afişarea indicelui respectiv, între `gsave` şi `grestore` (care salvează şi restaurează contextul grafic *curent*, încât modificările din interior sunt „ascunse" faţă de exterior).

Însă dacă avem nevoie şi de alte tipuri de literă, sau de expresii matematice mai complicate, atunci în loc să ne jucăm ca mai sus cu un „font obişnuit", mai bine căutăm o metodă generală, pe care să o putem aplica într-un mod standard pentru oricare figură şi pentru oricare necesităţi de etichetare...

5 Încapsularea în figură a notaţiilor matematice

Bineînţeles că există o asemenea „metodă generală", pentru a integra în documente notaţia matematică, în toată complexitatea specifică acesteia şi cu asigurarea unei foarte bune calităţi tipografice: sistemul de tipografie digitală TEX, folosit curent în mediile ştiinţifice (sistem creat prin 1970-80, de către D. KNUTH).

Dar vrând să obţinem o figură „independentă", trebuie să combinăm cumva fişierul nostru '`grafic.eps`' (fără liniile experimentate în §4) cu etichetele matematice pe care le-am obţine folosind direct TeX.
Vom proceda cam aşa: creem un document „`.tex`" *minimal*, conţinând exprimarea TeX a etichetei dorite; compilăm cu `latex` acest fişier, obţinând fişierul corespunzător în formatul DVI; de aici, cu programul `dvips`, ajungem la formatul „`.eps`" al etichetei respective, fişier pe care îl vom putea „combina" apoi cu '`grafic.eps`'.
Bineînţeles, vom căuta să automatizăm această succesiune de operaţii, având în vedere şi faptul că sunt de montat mai multe etichete.

Instituim un şablon minimal de fişier LaTeX, '`mathLabels.tex`', în care să putem scrie expresia TeX a unei etichete:

```
\documentclass{article}
\usepackage{xcolor}   % în eventualitatea că vrem să colorăm o etichetă
\usepackage[mathscr]{euscript}   % majuscule caligrafice (existente şi în text)
\pagestyle{empty}
\begin{document}
$ $   % aici se va scrie eticheta (între caracterele „dolar")
\end{document}
```

Nu este mai convenabil, să scriem în acest fişier toate formulele: ar fi complicat de identificat zona din pagină asociată fiecăreia (şi de stabilit dimensiunile acesteia) în cadrul fişierului „`.eps`" final.

Constituim un fişier-text '`labels.txt`', conţinând etichetele noastre în notaţia matematică TeX (fiecare pe câte o linie), dar fără delimitatori '`$`':

```
K_t
K_{1-t}
C_t
\\color{red}\\mathscr{K}
\\mathscr{E}
\\color{blue}\\mathscr{P}
```

Am avut grijă să dublăm caracterul '*backslash*' (folosit de TeX pentru a desemna o comandă), fiindcă acest caracter are de obicei semnificaţie specială—care se evită prefixând cu '\'—în diverse limbaje pe care le-am putea folosi mai departe.

Formulăm acum un program *Bash*, '`buildEPS.sh`', prin care se preia câte o linie de text din fişierul `labels.txt`, se scrie textul respectiv în fişierul `mathLabels.tex` în locul rezervat etichetei (am ales să folosim **`sed`**, pentru aceasta), apoi se lansează compilatorul `latex` şi se transmite rezultatul obţinut astfel, programului `dvips`:

```
#!/bin/bash
labels=(`cat labels.txt`)
i=1
for lb in "${labels[@]}"
do
    sed -i "s/\$.*\$/\$$lb\$/g" mathLabels.tex
    latex mathLabels.tex
    dvips -E mathLabels.dvi -o Label$i.eps
    ((i++))
done
```

Putem folosi `awk` (de pe linia de comandă) pentru a extrage şi a afişa coordonatele colţurilor boxei care conţine desenul etichetei, din cadrul declaraţiei '`%%BoundingBox`' existente în fişierele „`Label*.eps`" rezultate prin execuţia scriptului Bash de mai sus:

```
awk '/%%BoundingBox:/ {print $2,$3,$4,$5,FILENAME}'  Label*.eps
148 655 162 665 Label1.eps  # produce eticheta K_t
148 654 172 665 Label2.eps  # K_{1-t}
148 655 161 665 Label3.eps  # C_t
148 656 158 665 Label4.eps  # ℜ
148 656 156 665 Label5.eps  # ℰ
148 656 156 665 Label6.eps  # ℘
```

Aceste coordonate ne sunt necesare pentru a stabili ce translaţie avem de făcut pentru a poziţiona eticheta respectivă în locul cuvenit în cadrul paginii produse de `grafic.eps`; de exemplu, `-148 -655 translate` va aduce prima etichetă în colţul stânga-jos al paginii care conţine graficul produs de `grafic.eps` şi rămâne de calculat (sau de estimat prin încercări) ce translaţie mai trebuie făcută, încât s-o aducem lângă focarul K_t.

În '`grafic.eps`' am înfiinţat deja operatori (`x1`, `y1`, etc.) care ne dau coordonatele punctelor pe care vrem acum să le etichetăm şi de aceea este firesc

să încercăm să încorporăm fişierele „`Label*.eps`” chiar în '`grafic.eps`' (în loc de a încorpora toate cele şapte fişiere EPS într-un fişier nou „`pack.eps`”, cum ne propusesem *iniţial* în §1).

Sunt de întâmpinat două probleme: cum obţinem în '`grafic.eps`' coordonatele colţurilor etichetelor (evitând să operăm direct cu valorile 148, 655, etc.); cum modelăm calculele necesare aducerii etichetelor în locurile cuvenite din figură, având în vedere că sistemul de coordonate în care este produsă figura diferă de cel în care sunt produse etichetele.

5.1 Citirea valorilor %%BoundingBox

Fiind create automat (la fel), fişierele `Label*.eps` au acelaşi „preambul” şi conţin `%%BoundingBox` pe a cincea linie:

```
%!PS-Adobe-2.0 EPSF-2.0
%%Creator: dvips(k) 5.997 Copyright 2017 Radical Eye Software
%%Title: mathLabels.dvi
%%CreationDate: Tue Jul 2 05:27:39 2019
%%BoundingBox: 148 655 162 665
```

Până la înregistrarea colţurilor etichetei apar în total 166 de caractere; folosim operatorul **`file`** pentru a deschide în vederea citirii fişierul respectiv, folosim **`setfileposition`** pentru a poziţiona citirea de la al 166-lea caracter şi „citim” de aici 15 caractere (câte 3 cifre de colţ şi 3 spaţii intermediare); apoi folosim **`( ) search`** pentru a depune pe stivă, pe rând, fiecare grup de câte 3 cifre consecutive, transformându-l imediat în număr întreg prin **`token`**:

```
/readBB {  % numele fişierului este preluat din stivă
    /flb exch (r) file def
    flb 166 setfileposition
    flb 15 string readstring pop
    flb closefile  % pstack % (şirul de 15 caractere citit)
    3 { ( ) search pop  % parcurge până la primul spaţiu şi încarcă pe stivă
        token pop  % converteşte şirul de cifre în număr întreg
        3 1 roll pop pop exch } repeat
    token pop exch pop  % pstack % (avem pe stivă cele 4 coordonate)
} bind def
```

Am folosit **`pop`** (de câteva ori), pentru a elimina din stivă informaţiile de care suntem siguri că nu avem nevoie; de exemplu, `readstring` şi `token` adaugă în stivă *`true`* sau *`false`*, semnalând că operaţia a reuşit sau nu, iar `search` adaugă în stivă şi partea din şir rămasă după subşirul căutat (şi am folosit `3 1 roll` pentru a aduce în vârful stivei subşirul de 3 cifre respectiv, eliminând apoi celelalte subşiruri produse de `search`).

Aplicăm procedura '`readBB`' fiecăruia dintre fişierele „`Label*.eps`”, folosind **`forall`**:

```
[(Label6.eps) (Label5.eps) (Label4.eps)
 (Label3.eps) (Label2.eps) (Label1.eps)] {
    readBB
    4 array astore   % tablou ("array") conţinând cele 4 coordonate
} forall   % pstack % (cele 6 tablouri de coordonate)
```

Prin **astore**, cele 4 valori rezultate în stivă (la fiecare iteraţie) sunt comasate într-un tablou (încât vom putea accesa mai comod, colţurile respective). De observat că am transmis numele fişierelor „în ordine inversă", începând cu '`Label6.eps`'; astfel, cele 6 tablouri de coordonate vor fi aşezate în stivă în ordinea firescă în care urmează să le prelucrăm, începând (în vârful stivei) cu cel corespunzător lui '`Label1.eps`'.

Desigur, includem aceste secvenţe de program în fişierul '`grafic.eps`' (undeva înainte de „programul principal"); putem constata rezultatul (*decomentând* `pstack` de după `forall`, pentru a afişa conţinutul final al stivei):

```
vb@Home:~/keps$ gs -q grafic.eps
[148 655 162 665]   % colţurile etichetei din Label1.eps
[148 654 172 665]
[148 655 161 665]
[148 656 158 665]
[148 656 156 665]
[148 656 156 665]   % colţurile etichetei din Label6.eps)
```

Aceste coordonate ne vor servi pentru a calcula dimensiunile boxelor (în vederea repoziţionării etichetelor); de exemplu, obţinem lăţimea boxei extrăgând cu **get** abscisele colţurilor de la bază şi scăzându-le, cum descriem în acest mic experiment:

```
vb@Home:~/keps$ gs -q
GS> [148 655 162 665]   % Lx Ly Ux Uy (colţurile boxei)
GS<1> dup pstack
[148 655 162 665]
[148 655 162 665]
GS<2> 2 get pstack   % abscisa colţului dreapta-jos (Ux)
162   % tabloul din vârful stivei este înlocuit cu valoarea extrasă
[148 655 162 665]
GS<2> exch pstack   % aduce tabloul iarăşi în vârful stivei
[148 655 162 665]
162
GS<2> 0 get sub =   % extrage Lx (=148), face scăderea şi afişează rezultatul
14   % Ux - Lx = 14 = lăţimea boxei
GS> quit
```

Putem reformula mai simplu, instituind o variabilă de memorie care să preia tabloul din vârful stivei:

```
vb@Home:~/keps$ gs -q
GS> [148 655 162 665]
GS<1> /BB exch def   % variabila 'BB' preia valoarea (tablou) din vârful stivei
GS> BB 2 get BB 0 get sub =   % obţine lăţimea boxei (14)
```

Este de subliniat că fiecare nouă invocare a variabilei denumite mai sus '`BB`' va produce ca valoare tabloul aflat la momentul respectiv în vârful stivei

(şi nu un acelaşi tablou), încât odată definită, vom putea folosi o asemenea variabilă pentru toate etichetele noastre.

5.2 Poziţionarea etichetelor pe figură

Mai sus am citit din fişierele „`Label*.eps`" şi după ce am extras ce ne trebuie, le-am închis (folosind `closefile`); dar ceea ce avem de făcut de fapt este să *încapsulăm* programele EPS respective în '`grafic.eps`'.
Pentru aceasta, în principiu, trebuie să folosim operatorul **`run`**; de exemplu, `(Label1.eps) run` încorporează şi execută instrucţiunile din '`Label1.eps`', „redesenând" eticheta respectivă.
Numai că executarea instrucţiunilor „străine" încorporate astfel—dacă este *permisă*—decurge de regulă după ce pagina *curentă* este ştearsă (ejectată) şi sistemul grafic este reiniţializat (încât „redesenarea" se face într-o pagină *nouă*, cu parametrii grafici prevăzuţi de programul încorporat).

Pentru a încorpora etichetele chiar în pagina produsă de '`grafic.eps`' (evitând ejectarea acesteia), *minimul* necesar constă în anularea efectului instrucţiunii **`showpage`**, redeclarând-o (din timp!) ca procedură „vidă":

```
/showpage {} def   % ar fi descărcat pagina curentă, reiniţializând contextul grafic
```

Să înfiinţăm întâi procedurile care decurg din §5.1 pentru a prelua din stivă tabloul `%%BoundingBox` şi a determina dimensiunile corespunzătoare:

```
/BB exch def   % preia din stivă tabloul colţurilor etichetei curente
/wb {BB 2 get BB 0 get sub} bind def   % width (lăţimea etichetei)
/hb {BB 3 get BB 1 get sub} bind def   % height (înălţimea etichetei)
```

În '`Label*.eps`' era implicat sistemul de coordonate standard, originea fiind în colţul stânga-jos, iar unitatea de măsură $1 \equiv 1bp = 1/72inch$; în '`grafic.eps`' originea fusese fixată la $1.5um$ de marginea stângă şi $1um$ de baza paginii, unde alesesem $1um = 60bp$.
Avem de ales între aceste două posibilităţi de lucru:

1. Întâi „importăm" etichetele şi le poziţionăm; apoi abia, executăm „programul principal" (începând cu scalarea şi translatarea sistemului de coordonate) din '`grafic.eps`'; pentru poziţionare vom avea nevoie doar de valoarea $1um$.

2. Întâi executăm „programul principal" (începând cu scalarea şi translatarea sistemului de coordonate) din '`grafic.eps`' şi după aceea, „importăm" şi poziţionăm etichetele.

Să observăm că pentru a doua variantă, calculele de poziţionare revin la cele din prima variantă dacă (după ce executăm „programul principal" din '`grafic.eps`') reconstituim sistemul de coordonate standard, făcând o „translaţie inversă" şi o „scalare inversă" prin instrucţiunile suplimentare `-1.5 -1 translate` şi `1 um div dup scale`. Desigur, ne putem scuti

de o asemenea „reconstituire” *ad-hoc*, ambalând „programul principal” din 'grafic.eps' între gsave şi grestore:

```
    /showpage {} def   % evită ejectarea paginii curente
    /um 72 def   % fixăm ca „unitate de măsură” 1 ≡ 72bp = 1 inch
gsave   % „programul principal” (produce figura geometrică)
    um um scale   % coordonatele vor fi amplificate cu 1um
    1.5 1 translate   % 1.5um spre dreapta, 1um în susul paginii
    .01 setlinewidth 1 0 0 setrgbcolor Kardioid stroke   % K
    .005 setlinewidth   0.1 setgray Ellipse stroke   % elipsa E
    0 0 1 setrgbcolor   .005 setlinewidth Parabola stroke   % P
    0 setlinewidth   other_lines stroke   % celelalte linii şi arce
grestore   % restaurează contextul grafic iniţial
```

Este importantă, alegerea unităţii de măsură pentru construcţia figurii; cu $60bp$ cât alesesem iniţial (în §3), etichetele care vor completa acum figura se dovedesc a fi prea mari; alegând ca unitate $1inch$, vom avea o „potrivire” mai bună între mărimea de caracter a etichetelor şi mărimea figurii.
Am „ascuns” faţă de restul programului, scalarea şi translatarea coordonatelor din blocul referit prin „programul principal” (introducându-l între gsave şi grestore), dar acum apare o situaţie aproape ironică: pentru poziţionarea etichetelor pe figură trebuie să ştim translaţia folosită la construcţia figurii (adică tocmai ceea ce—chipurile—„ascunsesem”).

Eticheta K_t (ca să ne referim la un exemplu concret) trebuie „adusă” din poziţia indicată în %%BoundingBox-ul corespunzător ei (obţinut de pe stivă în variabila BB înfiinţată mai sus), într-o poziţie din pagina figurii care să corespundă punctului de coordonate (x1, y1) – unde x1 şi y1 invocă procedurile înfiinţate în §3 pentru determinarea abscisei şi ordonatei punctului semicardioidei corespunzător unei anumite valori a parametrului din ecuaţiile acesteia. Dar coordonatele rezultate sunt „numere abstracte”; poziţia din pagină corespunzătoare acestora se obţine scalând coordonatele cu $1um$ şi translatând cu $1.5um$ pe orizontală şi cu $1um$ pe verticală.

Să desemnăm prin [Lx Ly Ux Uy] valorile obţinute în variabila BB; calculul de poziţionare a etichetei trebuie să decurgă astfel: translatăm eticheta *spre stânga* cu (Lx, Ly) (instrucţiunea ar fi: -Lx -Ly translate); efectuăm apoi o a doua translaţie, cu $((x1 + 1.5)um, (y1 + 1)um)$. Astfel, colţul stânga-jos al etichetei ajunge acum în poziţia din pagină corespunzătoare coordonatelor „abstracte” (x1, y1); rămâne de făcut o anumită *corecţie*, ţinând seama de lăţimea şi înălţimea etichetei respective (încât în final, eticheta să fie aşezată cumva *lângă* punct):

```
% calculul translaţiei etichetei, pe orizontală şi pe verticală
/Tx {1.5 add um mul BB 0 get sub}  bind def
        % (x + 1.5)*um - Lx (x fiind preluat din stivă)
/Ty {1 add um mul BB 1 get sub }  bind def
        % (y + 1)*um - Ly (y din stivă)

% încapsulează şi poziţionează etichetele
gsave x1 Tx  wb sub   % corecţie orizontală (-wb) pentru eticheta K_t
```

```
      y1 Ty  hb 0.75 mul sub  % corecţie verticală cu -hb*0.75
      translate  (Label1.eps) run  grestore
gsave x2 Tx  wb 10 div sub  % corecţie cu -0.1wb şi +0.1hb
      y2 Ty  hb 10 div add
      translate  (Label2.eps) run  grestore
gsave xC Tx  wb 2 div sub
      yC Ty  hb 4 div add
      translate  (Label3.eps) run  grestore
gsave 1.8 Tx   .5 sqrt Ty  translate
      (Label4.eps) run  grestore
gsave 1.06 Tx  .4 sqrt Ty  translate
      (Label5.eps) run  grestore
gsave 0.68 Tx   .3 sqrt Ty  translate
      (Label6.eps) run  grestore
```

Pentru etichetarea curbelor am ales (prin calcule simple şi încercări) abscisele 1.8, etc. şi ordonatele $\sqrt{0.5}$ etc., aplicându-le `Tx` şi `Ty` pentru a obţine poziţia din pagină corespunzătoare lor (şi nu am avut nevoie de „corecţii").

În final însă (după atâta muncă), am putea fi şi dezamăgiţi: `evince` (sau poate şi vreun alt „*Document Viewer*") nu mai reuşeşte să interpreteze fişierul `grafic.eps` rezultat mai sus; iar invocând direct *Ghostscript* (prin `gs grafic.eps`) am avut (într-un anumit moment intermediar) surpriza de a obţine o pagină care conţine numai figura produsă de "`grafic.eps`" (*nu* şi etichetele încapsulate)...
Pe undeva este normal să fie aşa: *este* periculos să permiţi includerea într-un fişier a altor fişiere care conţin instrucţiuni executabile („străine").
Totuşi, *GS* permite şi evitarea unor restricţii privitoare la fişiere şi fonturi, prin opţiunea `-dNOSAFER`, încât prin comanda `gs -dNOSAFER grafic.eps` obţinem vizualizarea pe ecran a figurii complete (incluzând şi etichetele).

Desigur, nu vom putea folosi '`grafic.eps`' decât *având* alături şi fişierele `Label*.eps`; putem evita acest inconvenient (rezultând şi alte avantaje) *sintetizând* fişierele respective într-un document PDF:

```
vb@Home:~/keps$ ps2pdf -dNOSAFER  grafic.eps  grafic.pdf
```

Pentru a converti din EPS în PDF, `ps2pdf` angajează de fapt interpretorul *GS*, încât putem beneficia ca şi mai sus, de opţiunea `-dNOSAFER`.

Fişierul obţinut `grafic.pdf` măsoară numai $14.9kB$ (în timp ce fişierele „`Label*.eps`" obţinute prin `dvips`—când am executat scriptul '`buildEPS.sh`'—totalizează aproape $200kB$); vizualizând de exemplu prin `evince`, obţinem o pagină care conţine figura noastră (cu tot cu etichete) în partea din stânga-jos a ei. Dar aceasta nu este chiar ce ne-am dorit: în `grafic.eps` (v. §3) stabilisem în declaraţia `%%BoundingBox` limitele boxei care conţine figura noastră - iar acum vedem că acestea nu au fost în final, respectate; vizualizând cumva ca text, fişierul `grafic.pdf` - găsim că pagina respectivă a fost setată cu `/MediaBox [0 0 595 842]` (ceea ce corespunde formatului de pagină „A4").

Cum am arătat deja în §3, putem folosi `gs` cu `-sDEVICE=bbox` pentru a găsi `%%BoundingBox` (trebuie recalculat, fiindcă acum avem altă unitate de măsură decât aveam în §3) şi apoi cu **`pdfwrite`** pentru a insera (prin mecanismul '`pdfmark`') declaraţia `/CropBox` corespunzătoare cu `%%BoundingBox`, în fişierul '`grafic.pdf`' rezultat mai sus; obţinem astfel, figura din §1 (mai puţin eticheta 'O', pe care am adăugat-o ulterior).

Desigur, scopul pentru care faci o figură sau alta, nu este (în general) acela de a o arăta pur şi simplu pe ecran, sau a o tipări pe hârtie – ci de a o integra într-un anumit text referitor de exemplu, la contextul matematic asociat figurii. Figura obţinută mai sus poate fi salvată de exemplu în `images/keps3.pdf` şi va putea fi inclusă într-un fişier LaTeX existent prin comanda `\includegraphics{images/keps3.pdf}` (prevăzând în preambul `\usepackage{graphicx}`; având `/CropBox`, compilatorul de LaTeX va putea să constituie boxa necesară figurii, în cadrul boxei pe care o asociază paginii).

6 Arcele de parabolă sunt curbe Bézier pătratice

În procedura `Parabola` din §3, am generat „punct cu punct" arcul de parabolă $\mathcal{P}$ dat de $y = \sqrt{1-x}$, $x \in [0,1]$: se unea punctul curent (iniţializat în vârful parabolei) cu cel corespunzător abscisei depuse pe stivă la iteraţia următoare (prin mecanismul lui `for`), coborând abscisa cu câte `0.01` (în §3.1 am arătat însă că iterarea cu paşi fracţionari ar trebui evitată); în final avem o aproximare cu 100 de segmente a lui $\mathcal{P}$ – în fond, un set de 100 de instrucţiuni `lineto` care prin operatorul `stroke` vor produce pixeli vecini sau puncte suficient de apropiate, formând (pe ecran, sau pe hârtie) o imagine grafică acceptabilă a lui $\mathcal{P}$.

Vom arăta mai încolo că putem obţine o imagine la fel de bună a lui $\mathcal{P}$ (şi de fapt, a oricărui arc de parabolă), folosind o singură instrucţiune `curveto` (în loc de 100 `lineto`); dar pentru aceasta trebuie găsite în prealabil, anumite „puncte de control".

Să notăm cu P şi Q capetele arcului $\mathcal{P}$ (în cazul nostru $P(0,1)$ şi $Q(1,0)$); urmează să determinăm „punctele de control".

Fie H *intersecţia tangentelor* în P şi Q, la $\mathcal{P}$. Justificarea alegerii în acest fel a unui prim punct de control decurge dintr-o metodă generală de construcţie punctuală a unei conice (v. ***Steiner**'s generation of a conic*), care în cazul nostru revine la această proprietate: *dacă punctele U şi V împart segmentele orientate $\overline{PH}$ şi $\overline{HQ}$ într-un acelaşi raport $t \in [0,1]$, atunci punctul care împarte $\overline{UV}$ în raportul t descrie (când t variază de la 0 la 1) un arc de parabolă, care este tangent dreptelor PH şi QH.*

Pentru demonstraţie, să considerăm afixele punctelor, folosind litere omonime. Punctul care împarte $\overline{UV}$ în raportul t este $z(t) = (1-t)\mathsf{U} + t\mathsf{V}$; dar - fiindcă U împarte $\overline{PH}$ în raportul t - avem $u = (1-t)\mathsf{P} + t\mathsf{H}$; analog,

$v = (1-t)\mathsf{H} + t\mathsf{Q}$. Rezultă:

$$z(t) = (1-t)^2\mathsf{P} + 2t(1-t)\mathsf{H} + t^2\mathsf{Q},\ t \in [0,1]$$

Deci (fiind date de o funcţie de gradul doi în t) punctele $z(t)$ se află pe o parabolă care trece prin $z(0) = \mathsf{P}$ şi $z(1) = \mathsf{Q}$.
Avem $z'(t) = -2(1-t)\mathsf{P} + 2(1-2t)\mathsf{H} + 2t\mathsf{Q}$; rezultă că $z'(0) = 2(\mathsf{H}-\mathsf{P})$ (adică tangenta în $z(0)$ are aceeaşi direcţie ca vectorul $\overline{PH}$) şi $z'(1) = 2(\mathsf{Q}-\mathsf{H})$; deci dreptele PH şi HQ sunt tangente parabolei, în P şi respectiv în Q.

Arcul de parabolă $z(t)$ rezultat mai sus este curba Bézier *pătratică* definită de capetele P, Q şi de punctul de control H; dacă am avea o instrucţiune (similară cu `lineto`) care să furnizeze această curbă Bézier (primind ca argument capetele şi punctul de control), atunci am putea trasa $\mathcal{P}$ folosind numai această instrucţiune. Însă pentru a viza şi alte curbe (nu numai parabola), *PostScript* prevede implicit folosirea curbelor Bézier *cubice*, care necesită câte *două* puncte de control; vom vedea mai jos că pentru cazul unui arc de parabolă, acestea pot fi determinate plecând de la punctul H definit mai sus.

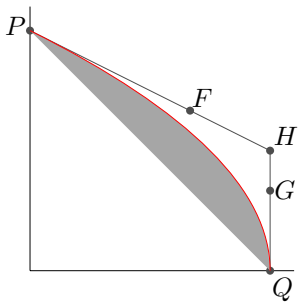
Să verificăm într-un caz concret, că pătratica Bézier definită de capetele unui arc de parabolă şi de intersecţia tangentelor acestuia în capete *coincide* cu arcul respectiv.
Pentru cazul nostru $\mathcal{P}:\ y^2 = 1-x,\ x \in [0,1]$ şi prin derivare avem $2yy' = -1$; deci $y'(0) = -\frac{1}{2y(0)} = -\frac{1}{2}$, încât ecuaţia tangentei în $P(0,1)$ este $y-1 = -\frac{1}{2}x$; tangenta în $Q(1,0)$ este $x = 1$. Rezultă prin intersecţie $H(1, \frac{1}{2})$ şi curba Bézier pătratică asociată mai sus este $z(t) = (1-t)^2 i + 2t(1-t)(1+\frac{1}{2}i) + t^2$, adică separând părţile (reală şi imaginară): $x(t) = 2t(1-t) + t^2 = t(2-t)$ şi $y(t) = (1-t)^2 + t(1-t) = 1-t$; eliminând parametrul t rezultă exact ecuaţia iniţială, $y^2 = 1-x$.

7 Arcele de parabolă, prin cubice Bézier

Am văzut mai sus că ecuaţia arcului de parabolă de capete P şi Q, dat fiind punctul („de control") H în care se intersectează tangentele în capete este (în planul complex) $z(t) = (1-t)^2\mathsf{P} + 2(1-t)t\mathsf{H} + t^2\mathsf{Q},\ t \in [0,1]$. Avem de găsit două puncte F şi G cu aceeaşi proprietate ca H - adică FP şi GQ să fie tangente parabolei; deci *F şi G trebuie căutate pe dreptele HP şi HQ.*

Pe de altă parte, vrem un polinom de gradul *trei* care să „reprezinte" cumva, arcul de parabolă $z(t)$; cubica respectivă ar trece prin capetele P şi Q (pentru $t = 0$ şi $t = 1$) dacă polinomul ar conţine monoamele $(1-t)^3\mathsf{P}$ şi $t^3\mathsf{Q}$, iar celelalte două monoame ar conţine $t(1-t)$ (încât ele să nu afecteze valorile în capetele $t \in \{0,1\}$). Observând că $(1-t) + t = 1$, putem satisface aceste cerinţe înmulţind $z(t)$ cu $(1-t)$ şi respectiv cu t şi adunând apoi rezultatele; rezultă astfel această rescriere:

$$z(t) = (1-t)^3\mathsf{P} + (1-t)^2 t(2\mathsf{H} + \mathsf{P}) + (1-t)t^2(2\mathsf{H} + \mathsf{Q}) + t^3\mathsf{Q}$$



P `moveto` F G Q **`curveto`**

Punctele de control căutate F şi G ar fi reprezentate de cei doi coeficienţi din mijloc, dar nu direct - fiindcă (de exemplu, pentru primul coeficient) punctul de afix $2\mathsf{H}+\mathsf{P}$ este *exterior* dreptei HP (originea planului fiind fixată arbitrar). Pentru ca F să se afle „la momentul" t pe segmentul orientat $\overline{PH}$ şi în acelaşi timp, să se afle pe raza polară a punctului $2\mathsf{H}+\mathsf{P}$ trebuie îndeplinită pentru un anumit factor λ, condiţia: $(1-t)\mathsf{P}+t\mathsf{H}=\lambda(2\mathsf{H}+\mathsf{P})$ din care deducem $2\lambda=t$ şi $\lambda=1-t$, care adunate, ne dau $\lambda=\frac{1}{3}$. Prin urmare $F=\frac{1}{3}(2\mathsf{H}+\mathsf{P})$; pentru celălalt coeficient este suficient să schimbăm P cu Q (dar putem şi repeta analog, calculul) şi avem $G=\frac{1}{3}(2\mathsf{H}+\mathsf{Q})$.

Pentru a evita factorul $\frac{1}{3}$, putem amplifica monoamele din mijloc cu 3, obţinând o expresie mai obişnuită a cubicei Bézier:

$$z(t)=(1-t)^3\,\mathsf{P}+3(1-t)^2t\,\mathsf{F}+3(1-t)t^2\,\mathsf{G}+t^3\,\mathsf{Q},\ t\in[0,1]$$

asociate capetelor unui arc de curbă date de coeficienţii extremi şi punctelor de control date de coeficienţii din mijloc.

8 Instrucţiunea `curveto`

Cubicele Bézier sunt modelate în *PostScript* prin operatorul `curveto`: acesta presupune că punctul curent (stabilit de exemplu printr-o instrucţiune `moveto`) este capătul de start al arcului şi cere de pe stivă coordonatele punctelor de control şi pe cele ale capătului final. De exemplu, pentru arcul de parabolă $\mathcal{P}$ avem din §6 $P(0,1)$, $Q(1,0)$ şi $H(1,0.5)$ iar după §7 găsim $\mathsf{F}=\frac{1}{3}(2\mathsf{H}+\mathsf{P})=(\frac{2}{3},\frac{2}{3})$ şi $\mathsf{G}=(1,\frac{1}{3})$; deci pentru a obţine $\mathcal{P}$ sunt suficiente aceste două instrucţiuni: `0 1 moveto` (care stabileşte P ca punct curent) şi apoi `0.66 0.66 1 0.33 1 0 curveto`; desigur că n-am introduce „0.66 0.66", ci am prefera '`2 3 div dup`'.

În programul următor, redefinim faţă de §3 '`Parabola`' (folosind aceeaşi metodă ca la definiţia '`Kardioid`' din §3.2); aceasta produce un contur poligonal pentru $\mathcal{P}$ cu N=100 segmente (instrucţiuni `lineto`). Adăugăm apoi o procedură care produce $\mathcal{P}$ cu o *singură* instrucţiune `curveto`; dar ne convingem,

folosind `pathforall` şi `flattenpath` că de fapt, această instrucţiune acoperă la execuţie (prin `stroke`) un anumit număr (dar mic, comparând cu N=100) de instrucţiuni `lineto`.

```
%!PS-Adobe-2.0 EPSF-2.0
/N 100 def
/Parabola {   % prin N segmente
    0 1 moveto
    0 1 N {N div dup 1 exch sub sqrt lineto} for   % i/N sqrt(1 - i/N)
} bind def

/Parabola3 { % drept cubică Bézier (o instrucţiune 'curveto')
    0 1 moveto
    2 3 div dup 1 1 3 div 1 0 curveto
} bind def

108 108 scale  1 2 translate  .004 setlinewidth

gsave  .65 setgray
    Parabola fill  % „umple" conturul poligonal (de N segmente)
grestore

gsave  1 0 0 setrgbcolor
    Parabola3  % prin 'curveto'
    {[3 1 roll] == } {} {[7 1 roll (curveto)] == } {} pathforall
    (\n) print
    flattenpath  % înlocuieşte 'curveto' prin secvenţe 'lineto' echivalente
    {[3 1 roll] == } {[3 1 roll (lineto)] == } {} {} pathforall
    stroke  % intern, 'stroke' va aplica 'flattenpath'
grestore
```

`stroke` (care controlează, în final, trasarea efectivă, pe ecran sau pe hârtie) înlocuieşte automat conturul indicat într-o instrucţiune `curveto`, printr-un contur („echivalent" cubicei Bézier iniţiale) constituit din instrucţiuni `lineto` – probabil chiar acelea pe care le-am cerut prin `pathforall` (după ce am liniarizat direct, invocând `flattenpath`) şi pe care le putem vedea executând programul prin *GS*:

```
vb@Home:~/keps$ gs -q  par_bez.eps
[-1.73843237e-05 0.999990702]  % 0 1 moveto
[0.666680694 0.66666311 1.00000823 0.333335608 1.00000823
    8.06229491e-06 (curveto)]
% flattenpath transformă conturul într-o secvenţă (echivalentă) de lineto:
[-1.73843237e-05 0.999990702]
[0.0593211092 0.970321417 (lineto)]
[0.234326661 0.874992847 (lineto)]
[0.437496513 0.749995053 (lineto)]
[0.609363139 0.624997199 (lineto)]
[0.749969602 0.499999374 (lineto)]
[0.859358788 0.37500155 (lineto)]
[0.937487841 0.250003725 (lineto)]
[0.984356642 0.125005886 (lineto)]
[1.00000823 0.0312682688 (lineto)]
```

```
[1.00000823 8.06229491e-06 (lineto)]
>>showpage, press <return> to continue<<
```

Dacă decupăm secvenţa celor 10 instrucţiuni '`lineto`' afişate şi o încorporăm (eliminând desigur, parantezele) la sfârşitul programului, putem constata că rezultă (suprapus) acelaşi contur ca şi cel produs de '`Parabola`' şi de '`Parabola3`'.

Este interesant de comparat: noi am generat segmentele (prin `for`) folosind (ceea ce este absolut uzual) un pas *constant* (aici, 0.01), fără a distinge în vreun fel punctele curbei; în schimb, `flattenpath` (sau `stroke`) - atunci când liniarizează conturul indicat în `curveto` - distinge cumva (ceea ce este desigur, mai *inteligent*), între părţile mai întinse şi cele mai bombate ale arcului, parcurgându-le cu paşi ceva mai mari, respectiv ceva mai mici (a vedea diferenţele absciselor la primele şi respectiv la ultimele instrucţiuni '`lineto`' dintre cele redate mai sus în `[…]`).

Pare deci *mai bine* (până la vreo probă contrarie) să calculăm punctele de control necesare şi să folosim `curveto`, decât să iterăm `lineto` pe 100 de paşi echidistanţi (şi de obicei, mai mulţi), trasând câte un segment de la punctul curent la cel următor; în cazul unei curbe mai „largi" sau mai complicate, ne putem gândi să o segmentăm întâi în câteva bucăţi şi să folosim `curveto` pentru fiecare dintre acestea.

8.1 Raţiunea de a fi a instrucţiunii `curveto` (litera 'S')

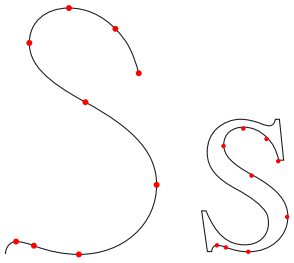
`lineto` şi `curveto` *nu* sunt instrumente de „grafică pe calculator" (precum de exemplu, funcţiile `plot()` din **R**) – ci sunt instrucţiuni proprii pe care PostScript *le foloseşte efectiv*, în scopul predeclarat de a formula o pagină astfel încât descrierea respectivă să poată fi interpretată la fel pe orice dispozitiv de redare a paginii.
Punctele sunt văzute ca nişte perechi de numere (şi nu există instrucţiuni pentru a le „trasa" ca atare); punctele vor putea arăta diferit de la un dispozitiv la altul, după cum este „pixelul" sau „punctul tipografic" al acestuia. Dar *ansamblul* de puncte constituit de un segment sau un arc Bézier va arăta „la fel", indiferent de dispozitiv sau scară – mai ales că transformările posibile nu se aplică figurii, ci sistemului de coordonate în care este exprimat conturul; tocmai de aceea, componentele paginii (inclusiv, caracterele) sunt descrise folosind `lineto` şi `curveto`.

Literele şi celelalte caractere sunt văzute ca fiind fiecare, un anumit *contur închis*, format din segmente şi curbe Bézier; astfel, ele îşi păstrează mai bine forma dacă se aplică scalări sau alte transformări – spre deosebire de cazul când ar fi reprezentate static, prin matrici predefinite de puncte.

Prin operatorul `charpath` putem obţine conturul existent pentru un caracter; de exemplu, `(S) false charpath` va adăuga conturului curent pe cel asociat pentru litera 'S' în cadrul fontului curent (dacă acest font nu este unul *protejat*). Iar dacă folosim apoi `pathforall` (cum am mai arătat), putem

afişa secvenţa de instrucţiuni care conturează litera respectivă; în următorul program am preluat din această secvenţă partea care corespunde conturului interior al lui 'S', în fontul `Times-Roman` (precizăm că în general, vom reda coordonatele din instrucţiunile `curveto` rotunjindu-le la a 4-a zecimală):

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 152 136 270 240

/Times–Roman findfont 18 scalefont setfont

144 144 translate  4 4 scale

% extragem dintr-o sesiune de lucru cu Ghostscript, conturul interior al lui 'S'
3.36217046 −0.464388192 moveto
3.3993 0.2043 3.8266 0.7059 4.36524916 0.705870092 curveto
4.7553 0.7059 5.3498 0.5758 6.0370 0.3158 curveto
7.3931 −0.2043 8.8605 −0.5015 10.2537 −0.5015 curveto
14.3960 −0.5015 17.5353 2.3033 17.5353 6.0185 curveto
17.5353 8.9348 15.5663 11.2010 10.8667 13.7645 curveto
7.1144 15.7892 5.6098 17.3495 5.6098 19.3185 curveto
5.6098 21.2689 7.1144 22.5878 9.3249 22.5878 curveto
10.9224 22.5878 12.4270 21.9191 13.6716 20.6374 curveto
14.7861 19.4857 15.2877 18.5755 15.8635 16.4765 curveto

21 0 moveto   % punctul de start pentru următorul contur
(S) false charpath   % adaugă conturului de mai sus, conturul literei 'S'

.1 setlinewidth  .1 setgray
stroke   % trasează efectiv cele două contururi

1 0 0 setrgbcolor
4.36524916 0.705870092 moveto   % punctează capătul final pentru prima 'curveto'
4.36524916 0.705870092 .2 0 360 arc fill

% capătul final pentru a doua instrucţiune 'curveto', a treia, etc.
```



Am marcat cu câte un mic disc roşu (produs prin `arc fill`) capătul *final* al fiecăreia dintre cele 9 bucăţi `curveto` ale conturului interior al literei (coordonatele cerculeţului sunt date de *ultimele două* valori din instrucţiunea `curveto` respectivă); acest contur este de două ori mai mare decât cel corespunzător din conturul literei redat în partea dreaptă a imaginii, fiindcă în sesiunea de lucru cu *GS* din care l-am preluat am folosit `36 scalefont`, în timp ce în programul de mai sus avem doar `18 scalefont`:

```
vb@Home:~/keps$ gs -q  % sesiune de lucru interactiv în Ghostscript
GS> /Times-Roman findfont 36 scalefont setfont
GS> 0 0 moveto
GS> (S) false charpath
GS> {[3 1 roll (moveto)] == } {[3 1 roll (lineto)] == }
GS<2> {[7 1 roll (curveto)] == } {(closepath) == }
GS<4> pathforall
[15.9749537 24.0367336 (moveto)]
[15.2133579 24.0367336 (lineto)]
[15.0648 23.2566 14.6747 22.8293 14.0802 22.8293 (curveto)]
...
[6.3157 0.7802 4.1052 2.7306 2.3219 7.0773 (curveto)]
[1.50461781 7.07727623 (lineto)]
[2.56342292 -0.464388192 (lineto)]
[3.36217046 -0.464388192 (lineto)]
[3.3993 0.2043 3.8266 0.7059 4.3652 0.7059 (curveto)]
...
[14.7861 19.4857 15.2877 18.5756 15.8635 16.4765 (curveto)]
[16.755127 16.4764938 (lineto)]
[15.9749537 24.0367336 (lineto)]
(closepath)
[20.015131 0.0 (moveto)]
GS>
```

Bineînţeles că am folosit `{[7 1 roll (curveto)] == }`, având în vedere că `curveto` are 6 parametri. Litera este un contur închis (se încheie prin `closepath`), constituit dintr-un sub-contur „exterior” (în cazul de faţă, cu 10 instrucţiuni `curveto`) şi unul „interior”, conectate şi completate prin câteva instrucţiuni `lineto`. În final, `20.015131 0.0 moveto` stabileşte „punctul curent” de la care să înceapă următoarea trasare de contur, dacă există (încât în programul de mai sus am putut estima prin `21 0 moveto`, unde să adaug conturul întreg al literei).

9 Aproximarea unei curbe prin cubice Bézier

Am ilustrat mai înainte faptul că *PostScript* conturează literele folosind secvenţe de instrucţiuni `curveto`; în treacăt fie zis – şi noi facem la fel în clasa *întâia*: desenăm literele cu „bastonaşe” şi arce de legătură (angajând intuitiv şi anumite „puncte de control”). Să vedem şi cum am putea folosi cubice Bézier pentru a contura o curbă sau alta, plecând de la ecuaţiile acesteia; subliniem că *nu* graficul ca atare, ne interesează (că atunci, l-am obţine fără bătaie de cap folosind **R**, sau vreun alt pachet de grafică); vrem să vedem cum ajungem la un program care să ne contureze curba respectivă, cât se poate de exact şi cu cât mai puţine instrucţiuni `curveto`.

Pentru cubica Bézier de capete P, Q şi cu punctele de control $F \equiv (2\mathsf{H}+\mathsf{P})/3$ şi $G \equiv (2\mathsf{H}+\mathsf{Q})/3$ – unde H era *intersecţia tangentelor în capete* - ajunsesem la exprimarea (literele drepte desemnând afixele punctelor omonime,

în $\mathbb{C}$):

$$z(t) = (1-t)^3\,\mathsf{P} + 3(1-t)^2 t\,\mathsf{F} + 3(1-t)t^2\,\mathsf{G} + t^3\,\mathsf{Q},\ t \in [0,1] \qquad (1)$$

Conturul asociat se obţine fixând „punctul curent" în P şi indicând operatorului `curveto` coordonatele punctelor F, G şi Q. Însă determinarea punctelor F şi G plecând de la intersecţia H a tangentelor în capete este incomodă.

Putem ocoli determinarea prealabilă a lui H, exprimând coeficienţii F şi G chiar din ecuaţia lui $z(t)$: *derivând*, găsim $z'(0) = -3\mathsf{P} + 3\mathsf{F}$ şi $z'(1) = -3\mathsf{G} + 3\mathsf{Q}$; deci

$$\mathsf{F} = \mathsf{P} + \frac{1}{3}z'(0), \quad \mathsf{G} = \mathsf{Q} - \frac{1}{3}z'(1) \qquad (2)$$

$z'(0)$ şi $z'(1)$ sunt pantele tangentelor curbei în capetele P şi Q (fiindcă avem $z(0) = \mathsf{P}$ şi $z(1) = \mathsf{Q}$). Pentru unele „cazuri speciale" trebuie totuşi să angajăm H; de exemplu, dacă $z'(1) = \infty$ şi $z'(0) \neq \infty$ atunci determinăm întâi intersecţia H a tangentei în P cu verticala prin Q şi apoi căutăm cumva G pe segmentul QH („teoretic", adică ignorând complet erorile de aproximare, avem $QG = \frac{2}{3}QH$).

Pentru a aproxima conturul unei curbe oarecare $\mathcal{C}$ (dată fiind, într-o formă sau alta, ecuaţia acesteia) putem proceda astfel: împărţim $\mathcal{C}$ în câteva părţi, având fiecare câte un capăt iniţial P şi unul final Q; determinăm valorile în P şi Q ale derivatei funcţiei asociate lui $\mathcal{C}$ şi găsim prin formulele (2) coeficienţii F şi G; „identificăm" arcul lui $\mathcal{C}$ delimitat de P şi Q prin cubica Bézier dată de P, F, G şi Q.

Desigur, „găsim F şi G *prin formulele (2)*" trebuie regândit, fiindcă P şi Q de pe curba $\mathcal{C}$ se obţin din ecuaţia acesteia nu pentru $t = 0$ şi $t = 1$, ci eventual pentru $t = \tau_0$ şi $t = \tau_1$ – indicând subintervalul parcurs de parametrul în funcţie de care se obţine porţiunea lui $\mathcal{C}$ de capete P şi Q.

Schimbarea de variabilă $t = \dfrac{\tau - \tau_0}{\tau_1 - \tau_0}$ transformă biunivoc $t \in [0,1]$ în $\tau \in [\tau_0, \tau_1]$. Pentru (2) nu interesează ce devine (1) prin această substituţie, ci doar cum se transformă $z'(t)$ trecând la noua variabilă τ; plecând de la regula de „derivare înlănţuită", avem: $\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{\mathrm{d}z}{\mathrm{d}\tau}\,\frac{\mathrm{d}\tau}{\mathrm{d}t} = (\tau_1 - \tau_0)\frac{\mathrm{d}z}{\mathrm{d}\tau}$. Aceasta înseamnă că în loc de $z'(t)$ trebuie să punem $h\,z'(\tau)$, notând $h = \tau_1 - \tau_0$; deci formulele (2) se adaptează la intervalul $[\tau_0, \tau_1]$ astfel:

$$\mathsf{F} = \mathsf{P} + \frac{h}{3}z'(\tau_0), \quad \mathsf{G} = \mathsf{Q} - \frac{h}{3}z'(\tau_1), \text{ unde } h = \tau_1 - \tau_0 \qquad (3)$$

Conturul lui $\mathcal{C}$ va fi constituit dintr-un număr de instrucţiuni `curveto` egal cu numărul părţilor considerate; pentru a creşte calitatea „identificării" cu arce Bézier, trebuie să folosim o partiţionare a lui $\mathcal{C}$ cu un număr suficient de mare de părţi (având în vedere eventual şi anumite „puncte speciale", cum ar fi vârfurile şi punctele de inflexiune).

10 Aproximarea semicardioidei prin cubice Bézier

Reluăm din §3.2 procedura `Kardioid`, prin care conturul semicardioidei superioare $\mathcal{K}$ (pe care acum, îl vom „umple" folosind `fill`) este constituit prin 1000 de instrucţiuni `lineto` (producând un grafic suficient de exact):

```
%!PS-Adobe-2.0 EPSF-2.0
% coordonatele punctelor semicardioidei K(t) (t este implicit, în vârful stivei)
/X {2 mul dup 1 sub mul} bind def   % X(t) = 2t(2t-1); Y(t) = 4t*sqrt(t(1-t))
/Y {dup dup 1 exch sub mul sqrt mul 4 mul} bind def

/N 1000 def   % aproximare poligonală (cu N segmente) a semicardioidei superioare
/Kardioid {
    0 0 moveto   % conturul pleacă din K(0) (nodul cardioidei)
    1 1 N { N div dup X exch Y lineto } for   % K(t)—K(t+1/N), t=i/N
} bind def

108 108 scale  2 1 translate
Kardioid   % instanţiază semicardioida (pentru noul sistem de coordonate)
.5 setgray  fill   % „închide" şi umple conturul cu pixeli gri
```

Dăm întâi un mic exemplu de constituire *manuală* (calculând direct, după formulele (3)), a unor instrucţiuni `curveto` care să „acopere" cumva, semicardioida umplută cu gri obţinută prin programul de mai sus (*vezi* figura de mai jos).

Cel mai simplu (pentru calcul manual - altfel, vom vedea că nu este deloc bine) ar fi să partiţionăm în *două* bucăţi: arcul descris pentru $t \in [0, \frac{1}{2}]$, având capetele $O(0,0)$ (nodul cardioidei) şi $B(0,1)$, respectiv pentru $t \in [\frac{1}{2}, 1]$ de capete $B(0,1)$ şi $A(2,0)$ (vârful cardioidei). Urmează să calculăm pentru fiecare arc, punctele F şi G din (3).

Amintim că punctele lui $\mathcal{K}$ sunt date de

$$x(t) = 2t(2t-1),\ y(t) = 4t\sqrt{t(1-t)},\ t \in [0,1]$$

şi găsim uşor

$$x'(t) = 2(4t-1),\ y'(t) = 2(3-4t)\sqrt{\frac{t}{1-t}} \quad (t \neq 1; y'(1) = \infty)$$

Pentru capătul O, avem: $z'_O = (x'(t), y'(t))|_{t=0} = (-2, 0)$ deci după (3), cu $h = \frac{1}{2}$, avem $\mathsf{F} = \mathsf{O} + \frac{h}{3} z'_O = (-\frac{1}{3}, 0)$.
Pentru capătul B, avem: $z'_B = (x'(t), y'(t))|_{t=0.5} = (2, 2)$ şi după (3) rezultă $\mathsf{G} = \mathsf{B} - \frac{h}{3} z'_B = (0,1) - \frac{1}{6}(2,2) = (-\frac{1}{3}, \frac{2}{3})$.

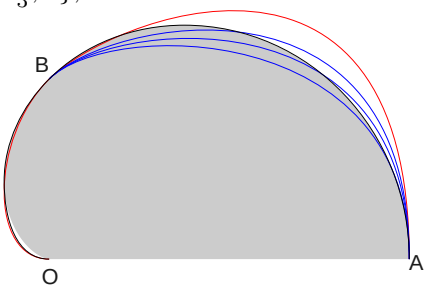
Prin urmare, cubica Bézier asociată primului arc va fi dată de `O(0,0) F(-1/3,0) G(-1/3,2/3) B(0,1) curveto`, ceea ce adăugăm în programul de mai sus astfel:

```
0 0 moveto  -1 3 div 0 -1 3 div 2 3 div 0 1 curveto
```

Pentru arcul lui $\mathcal{K}$ dintre B şi A putem determina $\mathsf{F} = \mathsf{B} + \frac{h}{3} z'_B = (\frac{1}{3}, \frac{4}{3})$, dar G *nu* mai poate fi obţinut prin (3) fiindcă $y'_A = y'(1) = \infty$; am dat de o *dilemă* tipică faţă de aplicarea formulelor (3) - cazul în care tangenta la curbă într-un capăt al arcului este verticală. Soluţia *ad-hoc* (dar nici nu ştim alta) ar consta în a alege G undeva pe tangenta în A, deci $G(2, \lambda)$; luăm $\lambda = 2$ şi adăugăm în program:

```
1 3 div 4 3 div   2 2   2 0   curveto
0 0 1 setrgbcolor   stroke   % cele două cubice sunt trasate cu roşu
```

Pentru comparaţie, am repetat ultima instrucţiune `curveto` pentru cazurile $\lambda \in \{2 - \frac{1}{3}, 2 - \frac{2}{3}, 1\}$, colorând acum cu albastru:



Contururile rezultate astfel (cel format din două arce roşu-roşu, sau cele constituite din arcul roşu şi unul dintre cele albastre) sunt departe de o aproximare acceptabilă a semicardioidei şi chiar este greu de crezut că separând în numai *două* arce am putea obţine una acceptabilă.

Să ştergem instrucţiunile `curveto` calculate manual mai sus şi să adoptăm acum ideea cea mai simplă: împărţim intervalul $[0, 1]$ în N bucăţi egale (cel puţin patru) şi pentru fiecare dintre acestea determinăm punctele de control necesare instrucţiunii `curveto` pentru aproximarea porţiunii corespunzătoare a lui $\mathcal{K}$. Vom scrie programul necesar pentru aceasta fără cine ştie ce elaborare, doar ca să vedem cam „cum s-ar face”; dar oricum am elabora, acest program va fi mult mai amplu (şi mai complicat) decât procedura iniţială `Kardioid` (care practic are o singură linie de program). Ce „deranja” la `Kardioid` era faptul că producea 1000 de instrucţiuni `lineto`; acum vom produce $\mathcal{K}$ într-o aproximare acceptabilă (dar nu aşa de bună ca folosind `Kardioid`) cu numai N=6 instrucţiuni `curveto` (sau mai bine, N=9 şi mai bine, N=12; cu `flattenpath` şi `pathforall` se va putea vedea că acestea acoperă maximum 50-60 de instrucţiuni `lineto`).

Introducem întâi procedurile `dX` şi `dY`, pentru a calcula $x'(t)$ şi $y'(t)$ (valoarea t fiind preluată de pe stiva operanzilor); apoi fixăm N (a încerca valorile 2..6..9; pentru N=10 sau N=11 programul eşuează, în urma propagării unor erori „de aproximare” - cum am mai observat în §3.1; pentru N=12 iarăşi „merge”). Introducem variabilele `h`=1/N şi `h3`=`h`/3, fiindcă aceste valori vor interveni frecvent în calcule; apoi iniţializăm `t0`, fixăm punctul curent în $O(0, 0)$ şi repetăm de N ori secvenţa de instrucţiuni care plecând de la capătul P≡`(x(t0), y(t0))`, determină Q≡`(x(t1), y(t1))` cu `t1 = t0 + h` şi

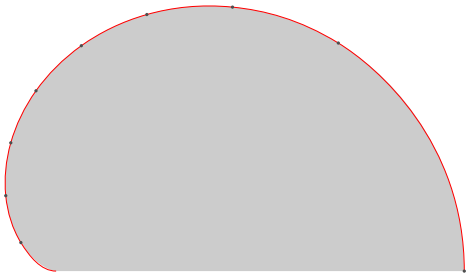
apoi calculează după formulele (3) punctele de control F şi G - producând instrucţiunea `curveto` pentru P, F, G şi Q.
Am întâmpinat faptul că $y'(1) = \infty$ folosind `ifelse`: când `t1` $\geq$ `0.99`, intersectăm tangenta în P cu verticala x=2 (tangenta în vârful cardioidei) găsind H la ordonata $y_H = y(t_0) + \frac{y'(t_0)}{x'(t_0)}(2 - x(t_0))$; punctul G are abscisa 2 şi ar trebui să aibă ordonata $\frac{2}{3}y_H$ - dar (cred că din cauza cumulării erorilor de aproximare) a trebuit să experimentez pentru a apropia mai mult cubica respectivă de cardioidă, mărind puţin ordonata lui G până la valoarea $\frac{2.6}{3}y_H$ (optând în final pentru 2.58/3):

```
/dX {8 mul 2 sub} bind def   % X'(t) = 8t - 2
/dY {/t exch def   % t == 6 8 t mul sub t 1 t sub div sqrt mul
} bind def   % Y'(t) = 2(3-4t)SQRT(t/(1-t))

/N 9 def
/h 1 N div def   % 1/N
/h3 h 3 div def   % h/3
/t0 0 def   t0 X t0 Y   moveto
N {
     /t1  t0  h add def   % t1 = t0 + h (t0 referă capătul iniţial al arcului curent)
     t0 X h3  t0 dX mul add   % F (primul punct de control)
     t0 Y h3  t0 dY mul add
     t1  0.99 ge   % la t1==1 avem tangentă verticală şi trebuie găsit H
    { 2 t0 Y   t0 dY t0 dX div   2 t0 X sub   mul   add
      2.58  3 div   mul }   % G este pe segmentul QH, încât QG/GH ≈ 2/3
    { t1 X h3 t1 dX mul sub   % G (al doilea punct de control) dacă t1 < 0.99
     t1 Y h3 t1 dY mul sub } ifelse
     t1 X t1 Y   % Q (capătul final pentru arcul curent)
     curveto
     /t0  t1  def   % t0 = t1
} repeat
1 0 0 setrgbcolor   0.005 setlinewidth   stroke
```

Am marcat şi punctele semicardioidei care sunt capetele finale ale celor 9 cubice Bézier determinate prin secvenţa de mai sus, adăugând în final:

```
/t0 0 def
0.3 setgray
N { /t1  t0  h add def   % t1 = i/N, i=1..N
     t1 X t1 Y moveto
     t1 X t1 Y .007 0 360 arc fill   % marchează punctul (x(t1), y(t1))
     /t0  t1  def
} repeat
```

În urma experienţei de lucru descrise mai sus devine totuşi clar, că mai totdeauna este de preferat un program simplu, modelând aproximarea poligonală obişnuită (fie şi cu 1000 de `lineto`, generate desigur printr-o instrucţiune repetitivă), unuia care constituie şi înlănţuie cubice Bézier care să aproximeze porţiuni ale curbei (mai ales dacă vrem o reprezentare cât mai exactă). `curveto` a fost introdusă nu pentru a trasa curbe pe baza ecuaţiilor acestora, ci mai degrabă pentru a permite realizarea *interactivă* (pe ecran, folosind mouse-ul)



aproximează $\mathcal{K}$ prin $N = 9$ cubice Bézier ($t = i/9$, $i = 1..9$)

de figuri (sau caractere) prin intermediul unor aplicaţii ca *Adobe Illustrator*, sau *CorelDraw*, *Inkscape*, etc.

11 Aproximarea semicardioidei folosind MetaPost

Care este (sau, cum determinăm) numărul minim de instrucţiuni `curveto` (cubice Bézier) pentru a reprezenta o curbă indicată (cu diferenţe imperceptibile la scară uzuală, faţă de graficul trasat - pe ecran sau pe hârtie - printr-un număr mare de puncte)?

Mai sus ajunsesem la o reprezentare cu 9 instrucţiuni `curveto` a lui $\mathcal{K}$, printr-un program *PostScript* generic (aplicabil *mutatis mutandis* şi altor curbe), care segmenta intervalul $[0, 1]$ în părţi egale şi determina pentru fiecare parte punctele de control necesare aproximării prin `curveto` a porţiunii corespunzătoare a lui $\mathcal{K}$; calculam punctele de control folosind derivatele în capetele subintervalului, lovindu-ne astfel şi de situaţia dilematică $y'(1) = \infty$ (când a trebuit să experimentăm asupra valorii „teoretice").

A devenit clar că este mai bine dacă segmentăm curba nu prin puncte legate arbitrar la valorile i/N, $i = 1..N$ ale parametrului t, ci prin unele puncte *caracteristice* ei (puncte de extrem, de inflexiune, etc.). Pe de altă parte, ne-a devenit clar că este mai avantajos să folosim *MetaPost*: operatorul desemnat prin '`..`' (denumit „*path_join*") creează o cubică Bézier care uneşte cele două puncte care (împreună eventual, cu *direcţia* tangentei) îi sunt indicate, angajând *intern* anumiţi algoritmi (destul de complicaţi pentru a ne feri să-i explicăm noi) de determinare a punctelor de control (spre deosebire de *PostScript*, în care avem doar posibilitatea elementară de a le specifica noi înşine).

Avem $x'(\frac{1}{4}) = 0$ (cu $y'(\frac{1}{4}) \neq 0$), deci în punctul $(x(\frac{1}{4}), y(\frac{1}{4})) = (-\frac{1}{4}, \frac{\sqrt{3}}{4})$ tangenta la $\mathcal{K}$ este verticală.
$y'(1)/x'(1) = \infty$, deci în $(x(1), y(1)) = (2, 0)$ tangenta este verticală.
$y'(0) = y'(\frac{3}{4}) = 0$, deci tangentele în $(0, 0)$ şi $(\frac{3}{4}, \frac{3\sqrt{3}}{4})$ sunt direcţionate orizontal.

Un alt punct important este $(x(\frac{1}{2}), y(\frac{1}{2}))$, fiindcă punctele lui $\mathcal{K}$ provenite din valori ale parametrului t simetrice faţă de mijlocul $\frac{1}{2}$ al intervalului $[0, 1]$

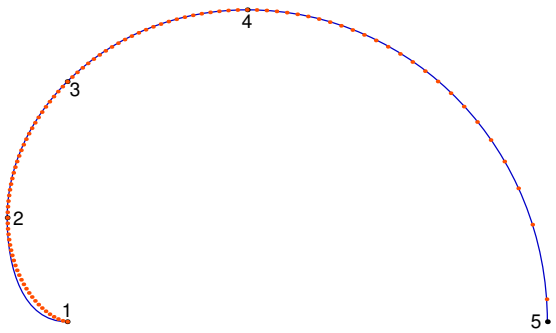
sunt legate (două câte două) prin proprietăţi semnificative (suma distanţelor la origine este egală cu 2; Oy bisectează unghiul razelor polare; etc.)
Avem $x'(\frac{1}{2}) = y'(\frac{1}{2}) = 2$, deci tangenta în $(x(\frac{1}{2}), y(\frac{1}{2}))$ este paralelă cu prima bisectoare a sistemului de axe.

Cele cinci „puncte importante" evidenţiate mai sus corespund împărţirii intervalului $[0, 1]$ în patru subintervale egale; dar acum am ajuns la ele plecând de la proprietăţi ale funcţiei (legate de direcţia tangentelor), în loc de a asuma din start ca în §10, o partiţionare echidistantă pe $[0, 1]$ (independentă de funcţie).
`z1`, ..., `z5` fiind punctele lui $\mathcal{K}$ rezultate pentru $t = \frac{i}{4}$, $i = 0..4$, putem formula conturul `z1{left}..z2{up}..z3{dir 45}..z4{right}..{down}z5`, însemnând în *MetaPost* constituirea şi îmbinarea a patru cubice Bézier: prima pleacă orizontal spre stânga din `z1`, curbează după punctele sale de control şi ajunge vertical în `z2`; a doua continuă spre `z3`, ajungând aici la 45 de grade faţă de orizontală; ş.a.m.d.

Un program *MetaPost* prin care să evidenţiem acest contur şi deasemenea (pentru comparaţie) conturul obţinut prin marcarea unui număr suficient de mare de puncte ale lui $\mathcal{K}$, se poate formula într-un fişier „`smcd.mp`" ca definiţie de figură `beginfig(1)`; ... `endfig`. Dar în principiu, *în prealabil* avem de setat anumite aspecte grafice (*fontul* de utilizat pentru etichetare, o *scalare* şi *poziţionare* în pagină convenabile, etc.) şi de formulat subrutine generice sau `macro`-uri de care vom avea nevoie în diverse definiţii de figură (astfel, `smc_point`(t) modelează ecuaţiile lui $\mathcal{K}$; `scardioid` foloseşte `smc_point` pentru a produce o variabilă de tip '`picture`' care la instanţiere, va marca un anumit număr de puncte ale lui $\mathcal{K}$; `path_nodes` etichetează nodurile conturului primit ca argument, poziţionând eticheta pe direcţia normalei la contur în punctul respectiv):

```
outputtemplate := "%j-%c.eps";   % produce 'smcd-1.eps', 'smcd-2.eps', etc.
prologues := 3;   % asigură înglobarea fontului, în fişierul EPS produs
defaultfont := "phvr8r";   % fontul 'Helvetica' (vom eticheta nodurile conturului)
defaultscale := 1.2;   % măreşte caracterele de la 8 la 8*1.2 puncte tipografice
% măreşte de 200 de ori; mută originea spre dreapta cu 10bp şi în sus cu 100bp:
def scale_shift = scaled 200 shifted (10, 100) enddef;

vardef smc_point(expr t) =   % punctul semicardioidei asociat lui t∈[0,1]
    (2*t*(2*t-1), 4*t*sqrt(t*(1-t)))
enddef;
def scardioid =   % trasează semicardioida prin 100 de puncte (instrucţiuni 'lineto')
    image(
        pickup pencircle scaled 2.3;
        for t = 0 step .01 until 1:
            draw smc_point(t) scale_shift withcolor (1, 0.3, 0);
        endfor
    )   % variabilă de tip 'picture' (o vom putea instanţia în oricare figură)
enddef;

def path_nodes(expr q) =   % marchează nodurile conturului
```

```
    pickup pencircle scaled 3; pair Z;
    for t=0 upto length(q):
        Z := point t of q;
        draw Z;   % marchează nodul şi îl etichetează pe direcţia normalei la curbă
        label(decimal(t+1), 7 unitvector(direction t of q)
                                rotated(-90) shifted Z) withcolor
                                    .1 black;
    endfor
enddef;

beginfig(1);   path p;   % „conturul" este o variabilă de tip 'path'
    z1 = (0, 0);   % nodul cardioidei (tangentă orizontală)
    z2 = (-1/4, sqrt(3)/4);   % cel mai din stânga punct (cu tangentă verticală)
    z3 = (0, 1);   % pentru t=0.5; tangenta este paralelă cu prima bisectoare
    z4 = (3/4, (3*sqrt(3))/4);   % cel mai de sus punct (tangentă orizontală)
    z5 = (2, 0);   % vârful cardioidei (tangentă verticală)
    p = ( z1{left}..z2{up}..z3{dir 45}..z4{right}..{down}z5 )
        scale_shift;
    pickup pencircle scaled 0.8;   % grosimea peniţei
    draw p withcolor .78 blue;   % cele 4 cubice Bézier (cu nuanţă de albastru)
    path_nodes(p);   % marchează şi etichetează capetele arcelor Bézier
    draw scardioid;   % instanţiază semicardioida conturată prin 'lineto'
endfig;
```



Invocând compilatorul de *MetaPost*, prin `mpost smcd.mp`, obţinem fişierul '`smcd-1.eps`' reflectat (printr-un „*Document Viewer*", ca `evince`) în imaginea de mai sus; deschizând într-un editor de cod-sursă (ca `gedit`), putem identifica uşor liniile asociate în *PostScript* (prin compilarea programului de mai sus cu `mpost`) celor patru operatori '`..`':

```
newpath 10 100 moveto
-26.74011 100  -40 144.2627  -40 186.60278 curveto
-40 229.53186  -20.38635 269.61365  10 300 curveto
49.55688 339.55688  104.02466 359.80835  160 359.80835 curveto
299.88953 359.80835  410 241.73889  410 100 curveto stroke
```

Dacă am înscrie deasupra acestor 5 linii `%!PS` (şi apoi linia `50 100 translate`, având în vedere că prima instrucţiune `curveto` are punctul final la abscisa `-40`) şi am salva fişierul ca „`fig-1.ps`", atunci deschizându-l în `evince` vom regăsi graficul celor 4 arce *1-2-3-4-5* din imaginea de mai sus.

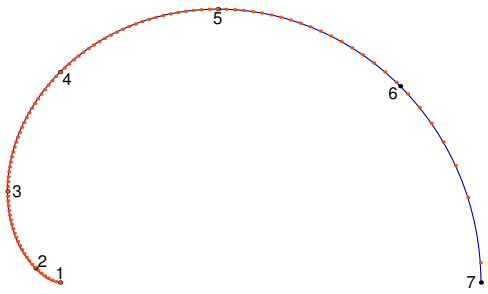
Acuma, să observăm că arcul albastru care uneşte punctele *1* şi *2* (reprezentând prima instrucţiune `curveto`) diferă *sensibil* faţă de porţiunea

(marcată prin puncte colorate „orange”) din $\mathcal{K}$ pe care voia să o aproximeze; deasemenea, mărind figura (spre 200%, prin operaţiile obişnuite într-un „*Document Viewer*”) putem sesiza că arcul albastru final *4–5* este puţin-puţin „deasupra” punctelor orange ale lui $\mathcal{K}$.

Este clar că pe $\mathcal{K}$ există un punct pe arcul *1–2* şi unul pe arcul *4–5* în care tangenta ajunge *paralelă cu a doua bisectoare* a sistemului de axe; pentru aceste puncte trebuie să avem $y'(t) = -x'(t)$, adică $(3-4t)\sqrt{t} = (1-4t)\sqrt{1-t}$.
Să observăm că această ecuaţie are „rădăcina *străină*” $\frac{1}{2}$ („străină” fiindcă duce la „1=-1”); ridicând la pătrat şi ţinând seama că astfel se introduce ca rădăcină şi cea „străină”, obţinem $32t^3 - 48t^2 + 18t - 1 = 0 \iff (2t-1)(16t^2 - 16t + 1) = 0$. Prin urmare, valorile t care ne dau punctele lui $\mathcal{K}$ în care tangenta are panta egală cu -1 sunt $t_{1,2} = \frac{1}{2} \pm \frac{\sqrt{3}}{4}$.

Adăugăm programului *MetaPost* de mai sus o a doua figură, considerând acum - prin `smc_point` ($t_{1,2}$) - şi cele două puncte ale lui $\mathcal{K}$ determinate mai sus:

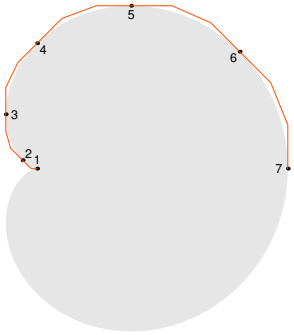
```
beginfig(2);
    numeric sq; sq = sqrt(3)/4;   % pentru a evita repetarea calculului
    z1 = (0, 0);
    z2 = smc_point(0.5 - sq);   % tangenta face 135 grade cu Ox
    z3 = (-0.25, sq);   % tangenta ajunge în poziţie verticală
    z4 = (0, 1);   % tangenta este înclinată cu 45 grade
    z5 = (0.75, 3*sq);   % tangentă orizontală
    z6 = smc_point(0.5 + sq);   % tangenta are direcţia egală cu (-45) grade
    z7 = (2, 0);
    path p, q;
    p := z1{left}..z2{dir 135}..z3{up}..z4{dir
        45}..z5{right}..z6{dir -45}..{down}z7;
    show p;   % afişează conturul constituit (variabila 'p')
    q := p scale_shift;   % scalează 'p', în vederea trasării efective
    pickup pencircle scaled 0.8; draw q withcolor .78blue;
    path_nodes(q);   % marchează şi etichetează capetele arcelor Bézier
    draw scardioid;   % instanţiază semicardioida conturată prin 'lineto'
endfig;

end   % pentru a evita modul de lucru interactiv („stop! NU mai am figuri/întrebări”)
```



De data aceasta, reprezentarea lui $\mathcal{K}$ (schiţat cu 100 de puncte orange) prin cele 6 cubice Bézier (colorate albastru) este destul de „fidelă”.

Fiind folosit **show p**, compilatorul ne dezvăluie în fişierul '**smcd.log**' constituţia conturului (şi am optat pentru valorile dinaintea scalării acestuia); să preluăm din acest fişier conturul **p** indicat şi să adăugăm (înaintea declaraţiei '**end**') o nouă figură, în care să întruchipăm întreaga cardioidă (simetrizând **p** faţă de Ox cu **reflectedabout** şi folosind apoi **buildcycle**) şi să trasăm conturul poligonal al punctelor de control pentru cele 6 cubice Bézier care formează semicardioida superioară:

```
beginfig(3);  % cardioida; poligonul punctelor de control
    path p, q, h, g, K;
    p = (0,0)..controls (-0.04654,0) and (-0.08301,0.03397)
 ..(-0.11603,0.06699)..controls (-0.21149,0.16245) and
     (-0.25,0.29778)
 ..(-0.25,0.43301)..controls (-0.25,0.64766) and
     (-0.15193,0.84807)
 ..(0,1)..controls (0.19778,1.19778) and (0.47012,1.29904)
 ..(0.75,1.29904)..controls (1.07574,1.29904) and (1.3856,1.16344)
 ..(1.61603,0.93301)..controls (1.86324,0.6858) and (2,0.34964)
 ..(2,0);  % semicardioida superioară (6 cubice Bézier)
    g = p reflectedabout((0,0), (2,0));  % semicardioida inferioară
    K = buildcycle(p, reverse g);  % „reuneşte", întruchipând cardioida
    fill (K scale_shift) withcolor .8white;
    q = p scale_shift;  % scaled 100 shifted (50, 300)
    path_nodes(q);
    h = point 0 of q for t=1 upto length(q):
        --postcontrol t-1 of q -- precontrol t of q endfor
        --point(length(q)) of q;  % conturul poligonal al punctelor de control
    draw h withpen pencircle scaled 0.75 withcolor (1, 0.3, 0);
endfig;
```



Arcul *1–2* având lungimea foarte mică, punctele de control ale sale n-au putut fi redate distinct (la scara folosită în program).

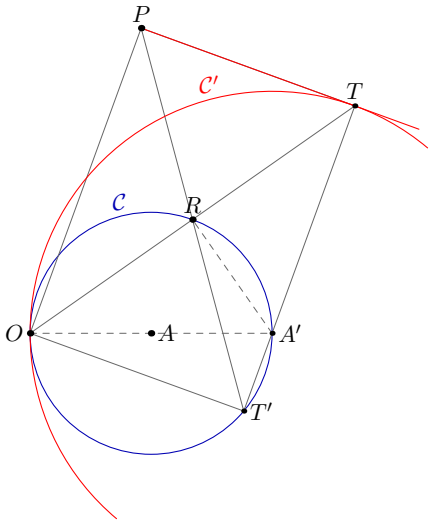
Punctele de control ale unei cubice Bézier se accesează prin **postcontrol** şi respectiv **precontrol** („post" vizează pe cel care urmează nodului iniţial al arcului, iar „pre" pe cel dinaintea nodului final). Punctele de control au fost astfel alese, încât segmentul care ar uni „precontrolul" unui arc cu „postcontrolul" arcului Bézier următor să fie *tangent* curbei pe care arcele respective o aproximează (cum se şi vede pe figură); de aici şi ideea aplicată mai sus: pentru a reprezenta o curbă cu un număr *cât mai mic* de cubice Bézier alegem cumva câteva puncte ale curbei în care cunoaştem direcţiile tangentelor

(şi montăm câte o instrucţiune `curveto` pentru fiecare două astfel de puncte, vecine la parcurgerea într-un acelaşi sens a curbei).
În *MetaPost* acest mecanism de calcul este implicit, cerându-se doar capetele arcului şi eventual, direcţia tangentei (la intrare sau/şi la ieşire) într-un capăt sau altul; el este *implicit* fiindcă nu se vizează o cubică Bézier ca atare (cum face `curveto` din *PostScript*), ci totdeauna, un *contur* format eventual (prin „*path_join*") din mai multe cubice Bézier înlănţuite („înlănţuirea" acestora necesită corelarea tangentelor, evidenţiată mai sus).

Anexă

Adăugăm câteva consideraţii elementare proprii, asupra cardioidei şi semi-cardioidei.

A De la cerc, la cardioidă



Fixăm punctele O şi A şi fie $\mathcal{C}$ cercul de centru A cu raza AO; notăm cu A' simetricul lui O faţă de A.

Fie R un punct oarecare pe $\mathcal{C}$ şi fie T simetricul lui O faţă de R, iar T' cealaltă intersecţie cu $\mathcal{C}$ a dreptei TA'; fie P simetricul lui T' faţă de R.

Ce curbe descriu T şi P, când R parcurge cercul $\mathcal{C}$?

Din construcţia indicată rezultă uşor că $OT'TP$ este un dreptunghi: $T'R$ este mediană în triunghiul dreptunghic $OT'T$ (avem $\measuredangle OT'A' = 90°$, fiindcă OA' este diametru în $\mathcal{C}$), deci $T'R = OR$ – prin urmare, diagonalele $T'P$ şi OT sunt egale.

În plus, $TA' = 2\,OA$ (fiindcă AR este linie mijlocie în $\Delta OA'T$, iar $AR = AO$); deci *locul lui T* (când R parcurge $\mathcal{C}$) *este cercul $\mathcal{C}'$* de centru A' şi rază $A'O = 2\,OA$.

Obs. $\mathcal{C}'$ provine din $\mathcal{C}$ prin omotetia de centru O şi raport 2, fiindcă pentru oricare puncte corespondente T şi R avem $\overrightarrow{OT} = 2\,\overrightarrow{OR}$.

Fiindcă $PT \perp TA'$ şi A' este centrul cercului $\mathcal{C}'$, rezultă că PT este tangentă în T la $\mathcal{C}'$; altfel spus, *P este proiecţia lui O pe o tangentă a cercului $\mathcal{C}'$.*

Pentru a găsi ecuaţia curbei descrise de P, să reperăm punctele faţă de originea $O(0,0)$ şi punctul unitar $A(1,0)$.

Cercul $\mathcal{C}'$ (cu centrul $A'(2,0)$ şi raza $OA'=2$) are ecuaţia $(x-2)^2+y^2=4$, care se mai scrie $x^2+y^2-4x=0$. Derivând în raport cu x, $2x+2yy'-4=0$, găsim panta tangentei în $T(\alpha,\beta)\in\mathcal{C}'$: $m=\frac{\mathrm{d}y}{\mathrm{d}x}\Big|_{x=\alpha}=\frac{2-\alpha}{\beta}$.

Deci ecuaţia tangentei la $\mathcal{C}'$ în $T(\alpha,\beta)$ este $y-\beta=\frac{2-\alpha}{\beta}(x-\alpha)$, care (ţinând cont imediat că avem $\alpha^2+\beta^2=4\alpha$) se reduce la

$$\alpha(x-2)+\beta y=2x \qquad (1)$$

Perpendiculara din $O(0,0)$ pe tangenta la $\mathcal{C}'$ în $T(\alpha,\beta)$ are ecuaţia $y=-\frac{1}{m}x$, sau

$$\alpha y-\beta x=2y \qquad (2)$$

Rezolvăm (cu „regula lui Cramer") sistemul de ecuaţii (1), (2) şi obţinem $\alpha=\frac{2(x^2+y^2)}{x^2+y^2-2x}$ şi $\beta=\frac{4y}{x^2+y^2-2x}$; înlocuind în $\alpha^2+\beta^2-4\alpha=0$, găsim ecuaţia locului intersecţiei P a tangentei în T la $\mathcal{C}'$ cu perpendiculara din O pe aceasta: $(x^2+y^2)^2+4y^2-2(x^2+y^2)(x^2+y^2-2x)=0$, sau într-o formă echivalentă

$$(x^2+y^2-2x)^2-4(x^2+y^2)=0 \qquad (3)$$

şi recunoaştem că aceasta reprezintă *cardioida* cu nodul $O(0,0)$ şi vârful $A'(2,0)$.

Obs. Locul proiecţiilor unui punct pe tangentele unei curbe date formează *podara* acesteia faţă de punctul respectiv; ceea ce am demonstrat mai sus revine la această proprietate binecunoscută: *podara unui cerc faţă de un punct al său este o cardioidă.* Podarele cercurilor (omotetice faţă de punctul comun O) $\mathcal{C}'$ şi $\mathcal{C}$ sunt cardioide omotetice, cu nodul comun O.

B Proprietăţile semicardioidei

Pentru $a>0$ considerăm cardioida $\mathcal{K}:(x^2-ax+y^2)^2-a^2(x^2+y^2)=0$ şi parabola $\mathcal{P}:y^2=a(a-x)$; notăm cu V vârful lui $\mathcal{P}$ şi cu M un punct cu abscisa între 0 şi a al lui $\mathcal{P}$. Să se arate că perpendiculara în M pe VM intersectează $\mathcal{K}$ în focarele unei elipse tangente în O axei Ox.
(*Problema* `L361`, Recreaţii Matematice nr. 1/2019)

Notând $x^2+y^2=at^2$ (unde $t>0$), $\mathcal{K}$ devine $(t^2-x)^2-at^2=0$, adică $(t^2-x-\sqrt{a}t)(t^2-x+\sqrt{a}t)=0$; dar posibilitatea $t^2-x+\sqrt{a}t=0$ trebuie exclusă, fiindcă ar conduce la $y^2\le 0$ (am găsi $y^2=at^2-x^2=at^2-(t^2+\sqrt{a}t)^2=-t^3(t+2\sqrt{a})$ şi cu $t>0$ am avea $y^2<0$, absurd).

Deci neapărat, $t^2-x-\sqrt{a}t=0$ - de unde $x=t(t-\sqrt{a})$; iar în acest caz, avem $y^2=at^2-x^2=at^2-t^2(t-\sqrt{a})^2=t^2\cdot t(2\sqrt{a}-t)$ şi $y^2\ge 0\Leftrightarrow t\in[0,2\sqrt{a}]$.

Prin urmare, cercurile $x^2 + y^2 = at^2$ taie graficul $\mathcal{K}$ în cel mult două puncte care sunt simetrice faţă de Ox, ceea ce înseamnă că punctele semicardioidei superioare pot fi exprimate în mod unic prin parametrul t:

$$x = t(t - \sqrt{a}), \quad y = t\sqrt{t(2\sqrt{a} - t)}; \quad t \in [0, 2\sqrt{a}]$$

Dar să eliminăm dependenţa între t şi a, scalând faţă de lungimea intervalului iniţial: $t = 2\sqrt{a}\,\lambda$, cu $\lambda \in [0, 1]$; noua parametrizare—renotând totuşi 'λ' cu 't'—este:

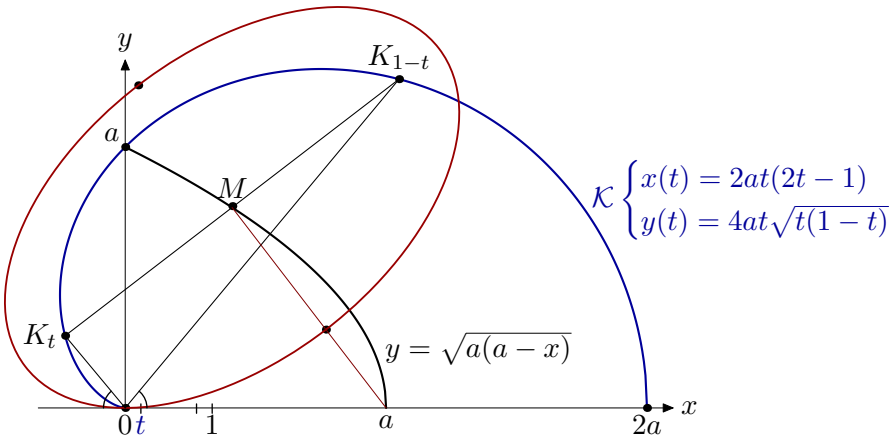
$$x = 2at(2t - 1), \quad y = 4at\sqrt{t(1 - t)}; \quad t \in [0, 1] \qquad (1)$$

Considerăm punctele K_t şi K_{1-t} provenite prin (1) din valori simetrice faţă de mijlocul $\frac{1}{2}$ al intervalului [0,1] pe care variază t; vom demonstra mai jos următoarele proprietăţi ale acestora:

(a) $OK_t + OK_{1-t} = 2a$;

(b) dreptele OK_t şi OK_{1-t} sunt egal înclinate faţă de axa orizontală;

(c) mijlocul segmentului K_tK_{1-t} aparţine arcului de parabolă dat de $y = \sqrt{a(a - x)}$, $x \in [0, a]$ iar mediatoarea acestui segment trece prin vârful parabolei $\mathcal{P}$.



Într-adevăr, din substituţiile iniţiale $x^2 + y^2 = at^2$ şi $t \equiv 2\sqrt{a}\,t$ rezultă că $OK_t = 2at$, deci $OK_t + OK_{1-t} = 2at + 2a(1 - t) = 2a$.

Panta dreptei OK_t este $2\frac{\sqrt{t(1-t)}}{2t-1}$ şi observând că transformarea $t \to 1-t$ (prin care ajungem de la K_t la K_{1-t}) schimbă doar semnul acestei expresii, deducem că suma pantelor dreptelor OK_t şi OK_{1-t} este zero, ceea ce este echivalent cu (b).

Mijlocul M_t al segmentului K_tK_{1-t} are coordonatele $x = a(2t - 1)^2 = a - 4at(1 - t)$ şi $y = 2a\sqrt{t(1 - t)}$ şi se vede imediat că satisfac $y^2 = a(a - x)$. Pe de altă parte, pentru distanţa de la vârful acestei parabole la punctul

K_t avem: $\mathsf{dist}^2((a,0),\,K_t) = (2at(2t-1)-a)^2 + 16a^2t^2t(1-t) = a^2(-4t^2+4t+1)$; observând acum că expresia $-4t^2+4t+1$ este invariantă faţă de transformarea $t \to 1-t$, deducem că $\mathsf{dist}((a,0),\,K_t) = \mathsf{dist}((a,0),\,K_{1-t})$, ceea ce înseamnă că vârful parabolei $\mathcal{P}$ se află pe mediatoarea segmentului K_tK_{1-t}.

Enunţul problemei rezultă interpretând proprietăţile arătate mai sus. (a) spune că K_t şi K_{1-t} sunt focarele unei elipse care trece prin O şi are axa mare de lungime $2a$; după (b), dreptele care unesc O cu focarele menţionate sunt egal înclinate faţă de Ox – deci Ox este tangentă în O elipsei. Conform (c), centrul acestei elipse aparţine parabolei $y^2 = a(a-x)$, iar axa mică a elipsei trece prin vârful $(a,0)$ al parabolei (şi are lungimea $MV = 4at(1-t)$ – ceea ce rezultă din triunghiul dreptunghic K_tMV în care avem $VK_t = a$, iar cateta MK_t se deduce plecând de la (1)).

Parabola $y^2 = a(a-x)$ *mediază* între arcele separate de Oy pe semicardioidă; cele câte două puncte astfel asociate sunt focarele unei elipse tangente în O axei Ox (iar panta axei focarelor este $\sqrt{\frac{b}{a}}$, unde $b = MV = 4at(1-t)$).

Obs. Ultima figură a fost produsă folosind *MetaPost* (şi apoi, `ps2pdf`), cu următorul „preambul”:

```
prologues:=3;
verbatimtex
    \documentclass[11pt]{article}
    \usepackage{lmodern}
    \usepackage{amssymb,amsmath}
    \begin{document}
etex
```

În baza acestui „preambul”, `mpost` (compilatorul de *MetaPost*) cu opţiunea `-tex=latex`, a „ştiut” singur cum să integreze în figură notaţia matematică LaTeX prevăzută (prin comenzi `label`) între `btex` şi `etex` (în timp ce lucrând în *PostScript*, trebuia pentru aceasta să procedăm cum am arătat în §5; şi `mpost` procedează cam tot ca în §5, dar „automat”, fără intervenţii suplimentare din afară). Din punct de vedere practic ar fi deci mai avantajos—pentru a realiza figuri geometrice adnotate matematic—să foloseşti *MetaPost*, în loc să foloseşti direct *PostScript* (doar că fişierul rezultat este de 3–4 ori mai mare).