A PLplot függvényábrázoló könyvtár

Elérkezett a C nyelven használható diagram varázslók ideje! Egy merész állítás bizonyítása következik.

Egész éjszaka számolt a gép – de az eredményt jelentő számsor önmagában nem mond túl sokat. Rengeteg munka volt az algoritmus megalkotása, viszont a látványra éhes kívülállókat a puszta számadatok nem győzik meg. Mit lehet ilyenkor tenni? Az egyik lehetséges megoldás szerint az eredménylistát bemásoljuk egy táblázatkezelőbe, majd meghívjuk azt a „tündért", vagy „varázslót" (mikor kit), aki másodpercek alatt mutatós grafikont állít elő adatainkból. Ha viszont erre a saját programunkból mi magunk is képesek vagyunk, miért választanánk a kerülőutat?

Kedves naplóm!

Valószínűségszámítást hallgatok az egyetemen. Az előadó szép számmal ad olyan feladatokat, melyek kísérletek ezerszer, vagy akár több tízezerszer történő elvégzésén és megfigyelésén alapulnak, és így számítógépes szimulációt igényelnek. Az eredményekből viszont csak úgy kaphatunk átfogó képet a megfigyelt jelenségekről, ha grafikusan is ábrázoljuk az eredményeket. Mivel nem voltam hajlandó föladni a *C* nyelv használatát, kénytelen voltam keresni egy olyan könyvtárat, amivel kedvenc programnyelvemből is készíthettem látványos bemutatót.

Igen egyszerű módszerrel álltam neki a keresésnek. Miután Debian Woody-t használok, elindítottam a `dselect` nevű csomagkezelő segédprogramot, majd leütöttem a / gombot és beírtam, hogy `plot`. Több alkalmazás nevében is szerepelt ez a szó, de csak egy olyat volt közöttük, amihez volt súgó csomag is. Azonnal telepítettem tehát a ***plplot***, ***plplot-dev***, és a ***plplot-doc*** csomagokat. Első lépésként a mellékelt példaprogramokat szerettem volna lefordítani, de erőfeszítésem kudarcba fulladt.

A *gcc* feloldhatatlan hivatkozások (undefined reference) tömkelege miatt dobott hibaüzeneteket. Ez természetes, gondoltam, hiszen nem adtam meg neki, hogy melyik függvénykönyvtárat kell hozzáfűzni (link) a programhoz. Póriasan fogalmazva nem mutattam meg, hogy melyik fájlban vannak a ***PLplot*** függvények. Vakon bízva a leírásban nekiveselkedtem a könyvtárhoz mellékelt 80 oldalas *.ps* dokumentumnak, ám ezzel kapcsolatban nem találtam semmit. Ekkor döntöttem úgy, hogy ellátogatok a ***PLplot*** hivatalos oldalára.

A ➲ *http://www.plplot.org/* oldalon ugyanazzal az állománynévvel már egy enyhén szólva kibővített, 160 oldalas leírás köszöntött, amit azon nyomban le is töltöttem. A kinyomtatott változat böngészése közben ráakadtam a megoldásra. A `plplot-config` szkript megfelelő módon történő meghívásával rövid úton letudhatjuk a fordító paraméterezését. Diadalittasan ütöttem be egyenként a szkript nevének betűit, de a gúnyosan villogó kurzor felett csak egy rideg elutasítás fogadott a ***Bash*** részéről: „`a parancs nem található`".

A megoldás küszöbén nem rettenthet el egy ilyen apróság, gondoltam, így letöltöttem a ***PLplot*** legfrissebb forrását. Az 5.3.1-es változat kitömörítése után a megszokott s

```
./configure
make
make install
```

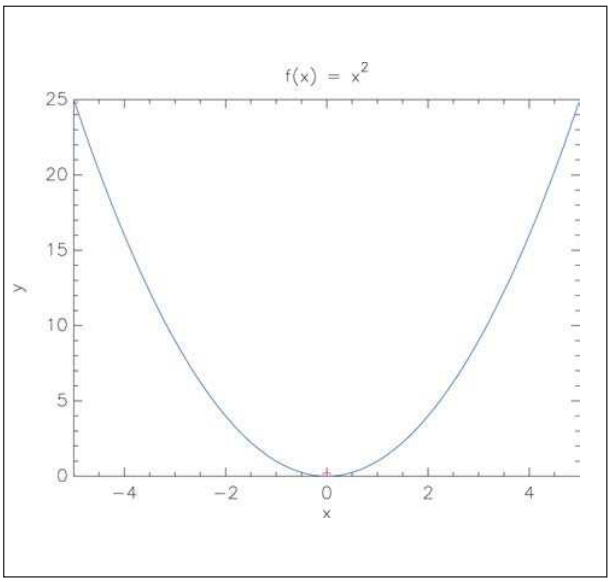
lépéseket követve kisvártatva egy működő ***PLplot*** változat büszke tulajdonosává váltam. Friss függvénykönyvtár, friss leírás és egy `plplot-config` parancsállomány fogadott a */usr/local* különböző alkönyvtáraiban. A példaprogramok már gond nélkül fordultak, így ezzel a fegyverarzenállal bátran vághattam neki a programfejlesztésnek.

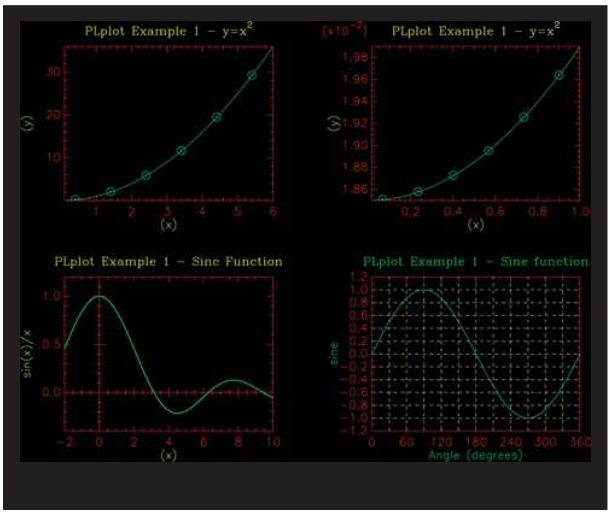
Egyfajta varázslat

Vágjunk bele a sűrűjébe! Az alábbi program egy közönséges parabolát ($y=x^2$ függvényt) ábrázol.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <plplot.h>

int main ()
{
    const PLFLT a = -5., b = 5., step = .1;
    const PLINT points = ceil ((b - a) / step) + 1;
    PLFLT *x, *y, t, min[2], max[2]; int i;

    if (NULL == (x = (PLFLT *) malloc (sizeof
      ➥(PLFLT) * points))) {
        perror ("malloc");
```

```
        return 1;
    }
    if (NULL == (y = (PLFLT *) malloc (sizeof
      ➥(PLFLT) * points))) {
        perror ("malloc");
        return 2;
    }

    min [0] = min [1] = max [0] = max [1] = 0;
    for (i = 0, t = a; t <= b; i ++, t += step) {
        x [i] = t; y [i] = pow (t, 2);
        if (min [1] > y [i]) {
            min [0] = x [i]; min [1] = y [i];
        }
        if (max [1] < y [i]) {
            max [0] = x [i]; max [1] = y [i];
        }
    }


/*****************************************************/
    plscol0 (0, 255, 255, 255);
    plscol0 (1, 0, 0, 0);
    plscol0 (15, 255, 0, 0);
    plinit ();
    plenv (a, b, min [1], max [1], 0, 0);
    pllab ("x", "y", "f(x) = x#u2#d");

    plcol0 (9);
    plline (points, x, y);

    plcol0 (15);
    plpoin (1, &min [0], &min [1], 0);
    plcol0 (15);
    plpoin (1, &max [0], &max [1], 0);

    plend ();

    free (y); free (x);
    return 0;
}
```





Mielőtt részletesen elemeznénk a program működését, próbáljuk ki! Indítás után azonnal egy kérdéssel szembesülünk:

```
Enter device number or keyword:
```

ENTER-t ütve rögtön szemünk elé tárul a gyönyörű grafikon, kék színnel jelezve magát a görbét. A legkisebb és a legnagyobb függvényérték egy-egy piros négyzettel van kiemelve. (Mint tudjuk, az x^2 függvénynek nincs maximuma, ezért őszintén remélem, hogy a matematika iránt olthatatlan szerelmet érző olvasók elnézik nekem ezt a fogalmazási pontatlanságot. Később visszatérek rá, hogy miért engedtem meg magamnak ezt az eléggé el nem ítélhető hanyagságot.)
Térjünk vissza a program által feltett kérdésre. Ez arra vonatkozik, hogy a ***PLplot*** különböző függvényein keresztül előállított ábra milyen eszköz segítségével kerüljön megjelenítésre. Az első lehetőség – amely egyben az alapértelmezett is – a grafikus felületen, vagyis egy X ablakban való megjelenítés. Ezen kívül a ***PLplot*** a legkülönfélébb eszközökhöz rendelkezik meghajtóval. Attól függően, hogy mit adtunk meg a fordítás során választhatunk ***Postscript***, ***Xfig*** és ***JPEG*** kimenet között.
Maga a program a beilleszthető fejlécállományok sorával kezdődik. A `stdlib.h` a dinamikus memóriakezelés, a `math.h` a hatványozást biztosító `pow()` függvény, míg a `plplot.h` a cikk tárgyát képező függvénykönyvtár definícióihoz szükséges. A `plplot.h` tartalmazza ezen kívül a `PLFLT` lebegőpontos, illetve a `PLINT` egész típusokat is, melyek a könyvtár használata esetén igen hatékonyan alkalmazhatók a számábrázoláshoz.
A program belépési pontja, amely jelen esetben az egyetlen függvény, a `main()`. Az első három sorban vezetjük be az új változókat. A ***PLplot*** segítségével X-Y diagramokat tudunk megjeleníteni. Az ábrázolás az a és `b` által meghatározott zárt intervallumban fog történni, az ábrázolási lépésközt pedig `step` jelzi. A `points` nevű

változóban tároljuk a ténylegesen kiszámítandó, és megjelenítendő pontok számát. Az összes többi pont számítása interpolációval történik.
Az X-Y diagram pontjai egy `x` és egy `y` nevű tömb elemei lesznek, természetesen úgy, hogy a két tömb azonos indexű tagjai alkotják egy pont koordinátáit. Bevezetünk egy `t` segédváltozót, egy-egy két elemű tömböt a minimum, és a maximum ábrázolásához, valamint egy `i` ciklusváltozót. Ez után két `malloc()` hívás következik, melyekkel területet foglalunk az `x` és az `y` tömbnek. Természetesen meg lehetett volna ezt oldani statikus tömbökkel is, ám ez rontana a program rugalmasságán.
A `min`, illetve `max` tömbök minden elemének 0 kezdőértéket adunk. Az ezt követő ciklusban számoljuk ki a görbe egyes diszkrét pontjait, illetve meghatározzuk a függvény minimumát és maximumát. Az `i` változót csupán az `x`, és `y` tömbök indexeléséhez használjuk, míg a `t` futó változó az intervallum pontjait veszi fel a megadott léptéknek megfelelően. A ciklus lényegét jelentő első sor a ciklusmagban magáért beszél. Az utána álló két elágazás egy megszokott módszer legalacsonyabb, illetve legmagasabb érték keresésére.
A megjegyzés után álló rész használja ki a ***PLplot*** függvénykönyvtár adta lehetőségeket. A ***PLplot*** biztosítja a programozót arról, hogy bármilyen eszközt is használ a megjelenítéshez, egy 16 színű paletta mindig rendelkezésére áll. Ennek a 0. színe jelenti az alapértelmezett hátteret, és az 1. az előteret. Ha ezeket nem módosítjuk, fekete háttér előtt piros színnel rajzol a könyvtár, ami a képernyőn még tűrhető, de nyomtatásra nem használható. Éppen ezért az első három sorban módosítjuk a palettát.
Nulladik színnek fehéret, elsőnek pedig feketét állítunk be (RGB megadás). Azért, hogy a piros továbbra is elérhető maradjon, beírjuk a 15. szín helyére. A `plinit()` függvényt minden ábrázolás előtt meg kell hívni, mivel bizonyos belső változóknak ez ad kezdőértéket. Lehetőségün van arra, hogy a `plinit()` meghívása előtt a programból meghatározzuk a kimeneti eszközt a `plsdev()` függvény segítségével. Ha ez elmarad, akkor a `plinit()` felkínál egy listát a felhasználónak az elérhető eszközökről, és az ő döntésén múlik, hogy az eredmény milyen formátumban jelenik meg.
A `plenv()` az ábrázolás környezetét állítja be. Első két paramétere az ***x***, második kettő az ***y*** tengelyen vett alsó és felső korlát. Az ***x*** tengely beállításához mi az a és `b` pontokat vettük, az ***y***-hoz pedig a számított minimum és maximum értékét. Többek között e paraméter miatt használtam az imént olyan pongyola módon a minimum és maximum fogalmát. A függvény ötödik paramétere adja meg, hogy a két tengely skálája azonos legyen-e (1), vagy sem (0).
Az utolsó paraméter a különböző címkék, tengelyek, rácsok megjelenítését állítja be. Mi itt csak egy dobozt kérünk a diagram köré, de az egyéb lehetőségekről bőven találunk leírást a dokumentációban.
A `pllab()` függvény segítségével határozhatunk meg címkéket az ***x*** és ***y*** tengelyhez, illetve itt adhatjuk meg a diagram címét. A karakterláncokat, amelyek különféle módosítókat is tartalmazhatnak, a jelzett sorrendben kell átadni,. Egy módosító mindig # karakterrel kezdődik, és hasonló beállításokra használható, mint a legtöbb szövegszerkesztő ***Formátum/Karakter...*** menüpontja. A `#u` az angol ***up (fel)*** jele, míg a `#d` a ***down (le)***. A bemutatott módszerrel a 2-es a felső indexbe kerül. Ha netán magára a # karakterre lenne szükségünk, azt a ## jelsorozattal jeleníthetnénk meg. Ha valamelyik címkét nem szeretnénk megadni, ott az üres karakterláncot kell átadni és nem ***NULL***-t!
Ezután jön a görbe megrajzolása. Előbb az előtérszínt állítjuk be kékre a `plcol0()` függvény segítségével. Az alapértelmezett palettában a kék a 9-es szín. Utána a `plline()` függvényt használjuk, mely két tömb megadott számú eleme alapján hoz létre egy képet. Első paraméterként átadjuk a pontok számát, második és harmadik paraméterként az `x` és `y` tömböket. Utóbbiak azonos indexű elemei alkotják egy pont koordinátáit. A függvény a szomszédos pontokat egy vonaldarabbal összeköti.
A maximum és minimum bejelölése is hasonlóan történik. Először beállítjuk pirosra az előtérszínt. Vegyük észre, hogy azért kapunk pirosat a 15-ös kiválasztásával, mert amikor az elején átrendeztük a palettát, ezt állítottuk be. A `plpoin()` hasonlít a `plline()`-hoz, de nem húzza össze a szomszédos pontokat és lehetőség van a pontok karakterének megválasztására. Első paramétere az ábrázolandó pontok száma, második és harmadik az x és y tömbök. Mivel egyetlen pontot ábrázolunk, kénytelenek voltunk a változó címét képezni, hogy hasonlítson egy tömbre. Az utolsó paraméterrel a 0-ás karaktert választjuk ki, ami épp egy négyzet.
Legvégül meghívjuk a `plend()` függvényt. Ez addig nem tér vissza, amíg a rajznak nincs vége. Ez egy ***Postscript*** dokumentum esetén a nyomtatás végét, egy ablak esetén az ablak bezárását jelenti. A `plend ()` visszatérésével a rajz megszűnik és a belső változók felszabadulnak. Mivel dinamikusan foglaltunk helyet az `x` és `y` tömböknek, ezeket is felszabadítjuk egy-egy `free()` hívással a program végén.
A `return 0;` utasítással az alkalmazás véget ér.

Hogyan tovább?

A fenti példaprogram amolyan állatorvosi ló, amin mindent ki lehet próbálni. Megpróbálkozhatunk például a ***sin(x)*** függvény ábrázolásával. Ehhez nem kell mást tennünk, mint a `for (;;)` ciklus magjában megfelelően módosítani azt sort, amely értéket ad az `y[i]` tömbelemeknek.
A ***PLplot*** természetesen sokkal többet tud annál, mint amit ebben a cikkben megmutattam. Nem csak ***C***-hez, de ***C++***-hoz, ***Fortranhoz***, ***Perlhez***, ***Tcl/Tk***-hoz, és ***Java***-hoz is van felülete, így szinte bárhol használható. Sőt, működik ***Microsoft Windows*** alatt is! A ***Visual Studio***-val könnyedén le lehet fordítani könyvtárat, akár ***DLL***-t is lehet belőle készíteni. Ez egyben azt is jelenti, hogy a ***PLplot***-tal készül programok forráskód szinten hordozhatóak. A könyvtár fejlesztése nem állt meg, folyamatosak a frissítések.
A ***PLplot*** lehetőséget biztosít oszlop-, torta-, és 3 dimenziós diagram létrehozására, vannak benne különböző vonal- és kitöltési stílusok, valamint különleges betűkészletek is. Akit érdekel a grafikus ábrázolás, mindenképpen töltse le a leírást, és szánjon rá egy-két délutánt az átolvasására. Megéri foglalkozni vele, mert egy idő után gyerekjáték a használata. Sok sikert hozzá!

Fülöp Balázs