Edícia vysokoškolských učebníc

Fakulta elektrotechniky a informatiky

Technická univerzita v Košiciach

ZÁKLADY PROGRAMOVANIA V PYLABE

Michal Kaukič

ZÁKLADY PROGRAMOVANIA V PYLABE

Edícia vysokoškolských učebníc FEI TU v Košiciach

Prvé vydanie 2006

Počet strán 59

Elektronická sadzba programom pdfTEX

Vytlačili Východoslovenské tlačiarne, a. s., Košice

ISBN 80-8073-634-0

Tabuľka 3: Iné funkcie pre grafiku a manipuláciu s ňou

`axes`	vytvorí nový súradnicový systém
`axhline`	nakreslí vodorovnú čiaru cez celý obrázok
`axvline`	nakreslí zvislú čiaru cez celý obrázok
`axhspan`	nakreslí vodorovný (vyplnený) stĺpec cez celý obrázok
`axvspan`	nakreslí zvislý (vyplnený) stĺpec cez celý obrázok
`bar`	urobí stĺpcový graf
`barh`	vodorovný stĺpcový graf
`boxplot`	obdĺžnikový a fúzatý graf stĺpcov dátovej matice
`clabel`	označovanie vrstevníc vo vrstevnicovom grafe
`colormaps`	`colormaps?` vypíše názvy farebných paliet
`delaxes`	vymaže daný súrad. systém z aktuálneho obrázka
`figlegend`	globálna legenda pre obrázok, nie pre súr. systém
`figtext`	pridá text v obrázkových súradniciach [0,1,0,1]
`hist`	kreslí histogram
`ioff, ion`	vypína a zapína interaktívny mod (efektivita!)
`imread`	načíta obrázok do číselného poľa
`imshow`	nakreslí obrázok z číselného poľa (viď `imread`)
`loglog`	graf s obidvomi mierkami na osiach logaritmickými
`matshow`	nakreslí maticu (veľkosti prvkov sú farebne odlíšené)
`pie`	koláčový graf, obľúbená to potrava manažérov
`plot_date`	ako `plot`, ale popisy osí sú dátumy
`quiver`	obrázok smerového poľa (diferenciálnej rovnice) alebo vektorového poľa
`rgrids`	prispôsobenie radiálnej sieťky a značkovania pre polárne súradnice
`scatter`	nakreslí roztrúsené body
`semilogx`	logaritmická mierka na x-ovej osi
`semilogy`	logaritmická mierka na y-ovej osi
`spy, spy2`	zobrazuje riedke matice (tam, kde sú nenulové prvky)
`stem`	„rastlinkový“ graf – od osi x idú steblá ku bodom
`table`	pridá tabuľku do obrázka
`thetagrids`	prispôsobenie uhlovej sieťky a značkovania pre polárne súradnice
`xlim, ylim`	nastavuje respektíve vracia hranice na osiach
`xticks`	nastavuje respektíve vracia popisy a značkovanie osi x
`yticks`	nastavuje respektíve vracia popisy a značkovanie osi y

Obsah

6 Záver

V tejto stručnej učebnici sme sa dotkli skutočne len niektorých oblastí, kde sa dá Pylab použiť. Takisto sme mohli spomenúť len málo z funkcií a možností, ktoré tento systém ponúka. Základom výpočtovej časti Pylabu je modul Scipy (OLIPHANT, 2004) a jeho submoduly. Z tých, o ktorých sme nehovorili, spomenieme teraz aspoň `stats` - modul pre štatistiku (obsahuje veľa rozdelení pravdepodobnosti a štatistických testov[4]).

Pylab má obrovský potenciál pre použitie nielen vo výučbe matematických a informatických predmetov, ale aj pre riešenie reálnych, rozsiahlych úloh inžinierskej praxe. Je to vďaka univerzálne použiteľnému programovaciemu jazyku Python. Keď budete dlhšie pracovať s Pylabom, určite si aj vy vytvoríte svoje vlastné minimoduly v Pythone, alebo budete používať aj ďalšie existujúce moduly, ktoré vám veľmi uľahčia prácu v špecializovaných aplikačných oblastiach.

Python nepodporuje symbolické manipulácie (teda napr. počítanie limít, derivácií, integrálov v tvare vzorcov). Existuje však systém `SAGE`, `http://sage.scipy.org/sage/` ktorý používa Python ako svoj hlavný programovací prostriedok a je určený na podporu výskumu a výučby v algebre, geometrii, teórii čísel, kryptografii, atď. Bližšie sa o tomto systéme môžete dozvedieť z dokumentácie (JOYNER A STEIN, 2006), ktorá je k dispozícii na vyššieuvedenej webovej stránke.

Na stránke `http://www.vrplumber.com/py3d.py` nájde čitateľ dobrý prehľad o aplikáciach a knižniciach, týkajúcich sa trojdimenzionálnej grafiky v Pythone. Autor zo svojej skúsenosti môže odporúčať napr. program `Mayavi`, `http://mayavi.sourceforge.net/`, čo je prezerač, umožňujúci interaktívnu manipuláciu s priestorovými objektami.

Dúfame, že aj táto učebnica podnieti ďalší záujem čitateľa o programovací jazyk Python a jeho početné rozširujúce moduly, ktoré môžu v mnohých prípadoch slúžiť ako rovnocenná náhrada komerčného softvéru, ba v mnohých ohľadoch (dostupnosť zdrojového kódu, široká a priateľská užívateľská komunita, otvorená aj pre začínajúcich programátorov, pravidelné konferencie tiež v Európe) ho aj predstihujú. Na serveri `www.py.cz` pre slovenských a českých užívateľov Pythonu nájdete veľa dokumentácie a materiálu, ktorý vám môže pomôcť v ďalšom raste a zdokonaľovaní sa v tejto oblasti.

[4]Existuje tiež modul `RPy`, ktorý umožňuje používať v Pythone, teda aj v Pylabe objekty a funkcie z programovacieho jazyka `R`, `http://www.r-project.org/`, čo je veľmi kvalitný Open Source systém pre štatistické výpočty (VENABLES A SMITH, 2006).

Predpokladáme, že tento text budete čítať pozorne a nad prečítaným sa budete aj zamýšľať, najlepšie s asistenciou počítača. Bez neho je to ťažké, niečo ako lízanie medu cez sklo. Nemusíte si čítať úplne všetko a „učiť sa to“. Najlepším učiteľom ja samotný Pylab, keď ho budete používať na riešenie problémov, ktoré vás bavia. Potom pre vás bude komunikácia s týmto systémom niečo ako rozhovor s inteligentným priateľom a pomocníkom.

Náš názor je, že **na matematických predmetoch sa neučíme programovať**. Predpokladáme, že máte aspoň základné návyky v algoritmizácii úloh (z oblasti aplikovanej matematiky alebo z inžinierskej praxe). Ak nie, možno vás k ďalšiemu štúdiu programovacích prostriedkov inšpiruje práve táto učebnica. Preto sme ju písali skôr formou konkrétnych príkladov (kde sa dalo, aj ilustrovaných grafikou) než suchopárnych programátorskych pravidiel. Dúfame, že si každý z vás nájde niečo, čo ho aspoň trochu zaujme a vyprovokuje k samostatnej činnosti a experimentovaniu.

Ak narazíte na problémy, snažte sa ich najskôr vyriešiť sami (veď na čo človek príde sám, to si aj najlepšie zapamätá). Keď to už nijako nejde, hľadajte pomoc vo svojom okolí (vrátane nás, učiteľov). Ak vieme, radi poradíme. Ak nevieme, aspoň máte lepší pocit, že nie ste sami a spolu budeme hľadať niekoho, kto by to vedieť mohol (aj keby bol u protinožcov).

Tento text je mienený ako úvod do práce s Pylabom, má vám teda **pomôcť začať pracovať** v tomto systéme. Na mnohé otázky v ňom nenájdete odpoveď. Ak však budú niektoré vaše otázky (alebo aj typické chyby a problémy) veľmi časté, radi tento text upravíme, či rozšírime o odpovede na ne. Pripomienky môžete posielať na adresu autora mike@frcatel.fri.utc.sk.

Nemohli ste si nevšimnúť, že je čoraz ťažšie zaobísť sa bez znalosti jazykov, angličtiny zvlášť. Ak máte v tejto oblasti slabiny, unikajú vám mnohé príležitosti. Zastávame názor, že učenie sa jazykov nie je nejaká separovaná činnosť (drvenie slovíčiek, gramatických poučiek a iné umelo vymyslené aktivity), ale treba ju podľa možností čo najskôr „zabudovať“ do našej bežnej práce. Samozrejme, ako štartovací bod je nutná istá minimálna slovná zásoba a základná znalosť gramatiky.

Angličtina je tiež v prípade Pylabu viac ako užitočná. Kto o tom ešte pochybuje, dúfame, že aj pri práci s Pylabom nájde ďalší podnet (a príležitosť) na zdokonalenie sa v oblasti jazykov. Pylab má vynikajúci systém na nápovedu, ale je to všetko v angličtine a v dohľadnom čase to nebude inak. Myslíme si tiež, že niektoré pojmy v oblasti informatiky strácajú prekladom na zrozumiteľnosti, takže aj k nami pridaným častiam systému sme písali nápovedu po anglicky.

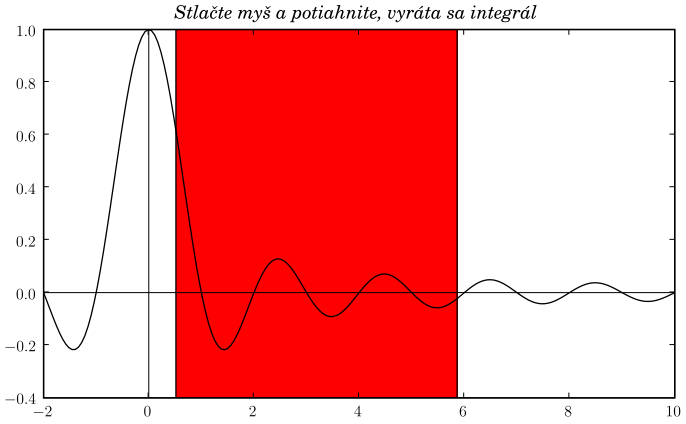
Ďalej nastavíme, aby sa pre výpisy matematických vzorcov používal TEX, nakreslíme integrovanú funkciu a súradnicové osi pre $x \in \langle -2, 10 \rangle$, urobíme vysvetľujúci text (titulok) obrázka (diakritické znamienka sa zadávajú ako v TEX-u).

```
rc('text', usetex=True)
fig=figure()                    # nove graficke okno
ax =gca()                       # aktualne osi do premennej ax

x=linspace(-2,10,200)
y=sinc(x); xlim(-2,10)          # hranice pre x
l=plot(x,y,'k')                 # graf integrovanej funkcie
axhline(0,color='k',lw=0.5)     # cierne tenke ciary
axvline(0,color='k',lw=0.5)     # pre surad. osi

t=title(r'\it Stla\v cte my\v s a potiahnite,\
        vyr\'ata sa integr\'al')
```

Nasleduje definícia obslužnej funkcie `onselect` pre `SpanSelector`. Jej parametre sú hranice `vmin, vmax`, vybrané pomocou stlačenia a pohybu ľavého tlačidla myši. Po výbere hraníc vyzerá grafické okno asi takto:



Tieto hranice sa využijú ako vstup do funkcie `quad` na numerickú integráciu. Oblasť v zvolených hraniciach, ohraničenú osou x a integrovanou funkciou, znázorníme pomocou farebného vyplnenia zelenou farbou a vypíšeme informácie o hodnote integrálu tak, ako je to na obrázku na predchádzajúcej strane. Takže dokončenie súboru `definteg.py` tvoria tieto príkazy:

1 Základný popis systému Pylab

1.1 Všeobecná charakteristika systému

Pylab, ako asi uhádnete, je skratka z Python Laboratory. Keďže inšpiráciou bol MATLAB, je to predovšetkým maticové laboratórium. Teda, interaktívny, maticovo orientovaný systém na vedecké a inžinierske výpočty a vizualizáciu dát (tzn. výsledky výpočtov môžete graficky prezentovať v rôznych typoch grafov, prípadne formou animácie).

Zdrojové programy, napísané v Pylabe (čo je vlastne Python s vhodnými knižnicami/modulmi) sa dajú bezo zmeny prenášať medzi jednotlivými implementáciami na rôznych počítačoch s rôznymi operačnými systémami. Špeciálne pre MS Windows sú vymyslené prostriedky (napr. `py2exe`, `http://www.py2exe.org/`) ako vašu aplikáciu (aj so všetkým, čo je pre jej beh potrebné, vrátane Pythonu a príslušných modulov) zbaliť do jedného vykonávateľného súboru. Takže, ak máte rozsiahlejší výpočet, môžete ísť na výkonnejší a spoľahlivejší počítač.

Asi je načase, aby ste sa posadili k počítaču a vyskúšali si zopár jednoduchých vecí. Hlavne, ako Pylab spustiť a ako z neho odísť. Pylab sa spustí príkazom `ipython -pylab`, ale to si pamätať nemusíte. V našich učebniach, kde budeme tento systém využívať, to bude jedna z položiek menu v grafickom prostredí XWindow, v ktorom sa ocitnete po prihlásení. Len tak mimochodom, budeme pracovať v OS Linux, ale to tiež nie je dôležité. Budeme využívať len Pylab a niekoľko málo iných aplikácii. Tí z vás, ktorí sa rozhodnú inštalovať a používať Pylab v MS Windows to budú mať podstatne zložitejšie.

Po spustení sa vám otvorí príkazové okno, v ktorom už budete môcť zadávať príkazy Pylabu a aj niektoré užitočné príkazy operačného systému (napr. `ls` – výpis adresára, `cd` – zmena adresára). Vaše vstupy sa značia na obrazovke ako `In [1]`, `In [2]`, ... a zodpovedajúce výstupy (odpovede) Pylabu sú `Out [1]`, `Out [2]`, ... Pre začiatok si môžete vyskúšať napr. `pylab?`, čo je prehľadná nápoveda o Pylabe. Keď sa chcete dostať znova na príkazový riadok, stlačte `q` (bude to vysvetlené o ďalej).

Práca v Pylabe sa ukončí príkazmi `quit` alebo `exit` alebo, čo je najpohodlnejšie, kombináciou klávesov `Ctrl-D` (to je znak konca súboru, v našom prípade signál na ukončenie vstupu zo štandardného vstupného súboru – klávesnice). Keď sa systém pýta, či chceme skutočne odísť, prikývneme (klávesnicou). Všetky príkazy, ktoré ste zadávali, sa zapisujú do súboru (konkrétne, je to súbor `history` v podpriečinku `.ipython` vášho domovského priečinka – to je ten, kde sa ocitnete po prihlásení a z ktorého nie je dôvod vôbec vyskakovať). Ak z Pylabu neodídete vyššie spomína-

```
    vc=connect('button_press_event',mk_point)        <-----
    show()

def mk_point(event):                                 <-----
    global vc, V, xmax
    if event.button==1:
        x,y=event.xdata,event.ydata
        if x>xmax:
            xmax=x
            plot([x],[y],'o')
            V.append((x,y))
    elif event.button==3:
        disconnect(vc)                               <-----
        save('Body.txt',V,fmt='%1.3f')
```

Interaktívne zadávanie bodov zaisťuje funkcia `Points_input()`. V nej je pomocou príkazov

```
axis([0,1,0,1])
ax=axes()
ax._autoscaleon=False
```

zaistené, že mierka na osiach bude stále taká istá a že sa nebude automaticky prispôsobovať súradniciam zadávaných bodov. Pre udalosti typu `'button_press_event'` je pomocou príkazu

```
vc=connect('button_press_event',mk_point)
```

registrovaná funkcia `mk_point`, ktorá ich bude obsluhovať. Ďalej je táto funkcia implementovaná.

Každá funkcia obsluhujúca udalosti musí mať len jediný parameter a to je (v submodule `matplotlib.backend_bases.Event`) objekt Pylabu `Event`. V obslužnej funkcii môžeme používať atribúty objektu `Event`, napr. tieto

`x`	x-súradnica v pixeloch od ľavej strany obrázku
`y`	y-súradnica v pixeloch od dolnej strany obrázku
`button`	stlačené tlačidlo myši (1, 2, 3 alebo None)
`inaxes`	objekt súrad. osí, ak je myš v tých súradniciach, alebo None
`xdata`	x-súrad. v užívateľských dátových súradniciach alebo None
`ydata`	y-súrad. v užívateľských dátových súradniciach alebo None
`key`	stlačený kláves, napr. 'a', 'b', '1'

Naša funkcia `mk_point` používa atribút `button` na zistenie, či je stlačené tlačidlo myši a ak je, tak ktoré je to. Myslíme si, že podľa tohto

ba celé balíky funkcií podľa potreby. Tí skúsenejší z vás môžu využívať v Pylab-e existujúce knižnice, napísané v C/C++ a FORTRAN-e. Takisto nie je problém čítať a zapisovať textové i binárne dáta do súborov.

Okrem širokej škály funkcií pre matice a vektory (napr. maticové operácie, jednoduchá manipulácia s indexami a submaticami, riešenie sústav lineárnych rovníc, inverzia matice, determinant, vlastné čísla a vlastné vektory, Fourierova transformácia, charakteristický polynóm, ...) máme v Pylabe tiež základné numerické algoritmy:

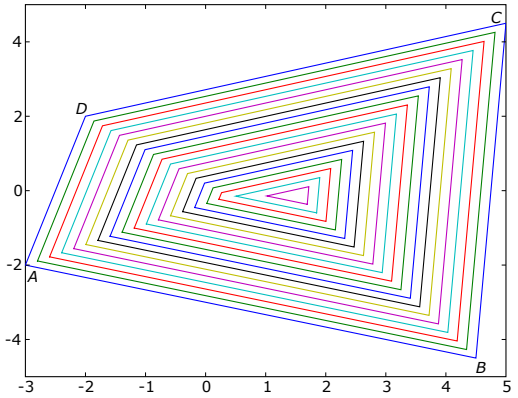
- interpolácia a aproximácia jedno- a dvojrozmerných dát (polynomická a splajnová),
- korene polynómov, riešenie nelineárnych rovníc a ich sústav,
- numerické integrovanie (aj pre dvojné a trojné integrály),
- optimalizácia funkcií viac premenných; existuje aj možnosť využitia Pythonu ako jazyka na modelovanie a riešenie optimalizačných úloh lineárneho a celočíselného programovania (napr. modul `Pulp`),
- riešenie diferenciálnych rovníc a ich sústav,
- funkcie pre pravdepodobnosť a štatistiku,
- špeciálne funkcie (ortogonálne polynómy, eliptické integrály, gama funkcia, Besselove funkcie, ...)
- funkcie na spracovanie signálu a obrazu.

Pritom si môžeme výsledky okamžite znázorniť pomocou grafov v pravouhlých alebo v polárnych súradniciach, prípadne parametricky. Máme aj špeciálne grafy (histogramy, koláčové grafy). Pylab dokáže tiež zobraziť vrstevnice plochy $z = f(x, y)$, príp. farebne vyplnený priestor medzi vrstevnicami (niečo ako mapa, kde farba závisí od nadmorskej výšky). Trojrozmerná grafika (kreslenie plôch) sa plánuje v novších verziách Pylabu.

V Pylabe môžeme vykonávať tiež ľubovoľné externé programy a príkazy operačného systému (napr. tak, že pred príkaz napíšeme výkričník), čo nám umožňuje využiť hocijaké naše už skompilované programy, ale aj iné programové systémy napr. na vytvorenie užívateľského rozhrania, generovanie dátových súborov pre Pylab, rozšírenie grafických možností Pylab-u, symbolické úpravy vzorcov (derivovanie, integrovanie, úprava algebraických výrazov napr. v systéme MAXIMA[3]).

[3] MAXIMA je kvalitný Open Source softvér pre symbolické výpočty, dá sa nájsť na adrese *http://maxima.sourceforge.net*. Užitočné je tiež užívateľsky príjemné rozhranie wxMaxima s domovskou stránkou *http://wxmaxima.sourceforge.net*.

Naznačíme si iný, geometricky názornejší spôsob riešenia príkladu o strelcoch. Uvažujme o vrstevniciach grafu funkcie $d(x, y)$ na štvoruholníku $\mathcal{S}$. Budú to menšie štvoruholníky, príp. degenerované na trojuholník alebo úsečku, ako to vidieť na obrázku. Graf funkcie $d(x, y)$ je plocha, pripomínajúca stoh slamy, vo všeobecnosti trochu asymetrický. Najvyšší bod toho stohu udáva nami hľadanú optimálnu polohu.



Vedeli by ste (samozrejme v Pylabe) tiež nakresliť podobný obrázok? Všimnite si, že vnútorné štvoruholníky, tvoriace vrstevnice, majú vrcholy na osiach uhlov pôvodného štvoruholníka (je to logické, lebo tam sú vzdialenosti od dvoch zo strán rovnaké).

Intuitívne sa zdá, že najlepšia bude taká poloha, že budeme mať od všetkých strán rovnakú vzdialenosť (a to čo najväčšiu). To by sme sa museli postaviť do stredu kružnice, opísanej štvoruholníku. Nie každý štvoruholník však takú kružnicu má (predstavme si napr. že je dlhý a tenký). Bude to asi tak, že hľadaný bod bude stred kružnice, dotýkajúcej sa zvnútra niektorých troch strán štvoruholníka. Alebo, inak povedané, priesečník osí dvoch vhodných vnútorných uhlov štvoruholníka (v našom konkrétnom štvoruholníku je to priesečník osí uhlov pri vrcholoch B a C). Z obrázka je to skoro jasné, skúste si to zdôvodniť presnejšie a realizujte tento postup programovo v Pylabe.

návať **maticové operácie a funkcie** (napr. výber submatíc a prvkov matíc a aj modifikáciu príslušných oblastí matice, vytváranie matíc z menších „blokov“, násobenie matíc, inverznú maticu, determinant matice, vlastné čísla a vlastné vektory a veľa iných vecí), nemusíme sa teda starať o „programovanie“ na úrovni prvkov matíc a vektorov s množstvom `for` cyklov a deklarácii. Tým sa zdrojový kód značne zmenší a sprehľadní.

Pokiaľ si však na taký (pohodlný, ale iný) štýl práce zvyknete, budete mať tendenciu aj jednoduché veci (z pohľadu Pylabu) robiť zložito, hlavne ak ste si predtým zvykli na „klasiku“ v PASCAL-e či C-čku. Zlaté pravidlo Pylabu hlása: **Vyhýbaj sa cyklom, kde sa to len dá.** Všetko sa snažte zapisovať aj vo vašich programoch štýlom, blízkym k obvyklému matematickému zápisu, ako píšete napr. na matematickej analýze či algebre.

Je dobre si uvedomiť, že vektor je špeciálnym prípadom matice (matica typu $(1 \times n)$ je riadkový a typu $(n \times 1)$ stĺpcový vektor). Ale tiež môžeme mať „jednorozmerný“ vektor, kde nám na riadkovosti či stĺpcovosti nezáleží a vtedy stačí zadať len jeden rozmer – počet prvkov.

Matica môže byť zadaná viacerými spôsobmi, napr.:

- pomocou príkazov a prostriedkov Pylabu (z klávesnice – zadaním prvkov matice alebo vytvorením matice v Pylab-skom programe)
- nahratím zo súboru (vytvoreného v Pylabe, C-čku, FORTRANe, exportovaného z databázy, príp. tabuľkového procesora alebo priamo vytvoreného v nejakom textovom editore).

Neodporúčame vám používať „školský“ spôsob interaktívneho zadávania rozmerov a prvkov matice z klávesnice (`Zadaj n: Zadaj A[1,1]` a pod.). Hodí sa to pre malé úlohy, ale s reálnym softvérom to nemá nič spoločné. Predpokladáme, že chcete byť skutoční profesionáli a nie „lepiči“, preto si trénujte správne programovacie návyky :-). Aj keď používate malé vstupné dáta, programujte tak, aby ste ten istý kód mohli použiť pre realistické, veľké vstupy. Zásadne používajte vstupné a výstupné (textové, pre rozsiahle dáta aj binárne) súbory, Python má veľmi príjemné prostriedky na prácu s nimi.

Budeme sa zaoberať najskôr priamym zadávaním matíc, teda prvým prípadom. Najjednoduchším spôsobom je **zadanie vymenovaním prvkov matice**. Niekoľko príkladov (skúšajte si „naživo“):

```
A = array([[1,2.5], [-1,3]]) # realna matica 2x2
B = array([-1,3,2**80])      # celocisel. jednorozmerny
                               # vektor, 2**80 je umocnovanie
Br= array([[1,-2,3]])        # riadkovy vektor
```

```
rr=rt[abs(rt.imag)<1.0e-8].real
```

pomocou ktorého jedným šmahom vyberieme len tie korene, ktoré majú zanedbateľnú veľkosť imaginárnej časti (prakticky povedané, sú to reálne korene) a potom ich zreálnime – zoberieme ich reálne časti, ktoré uložíme do číselného poľa `rr`. To potom usporiadame a máme hranice integrálu. Teraz už len použijeme funkciu `fsolve` na určenie koreňa nelineárnej rovnice $P(r) - 10 = 0$. Ako počiatočné priblíženie riešenia sme zobrali $r = 8$, teda takú dĺžku špagátu, že pastva bude určite neprázdna. Pre kontrolu, nám vyšla dĺžka špagátu 6.99133891149 m.

4.3 Minimalizácia funkcie dvoch premenných

Príklad 4.4 (Príklad o strelcoch). *Prenasleduje nás banda štyroch strelcov, pred ktorými sa skrývame v (oplotenom a vypuklom) štvoruholníku. Každý strelec sa pohybuje po jednej zo strán štvoruholníka a všetci majú rovnaký dostrel. Predpokladáme, že sa pohybujú (pre nich) optimálne, t. j. zaujmú na svojej strane vždy takú polohu, aby nám boli najbližšie. Akú si máme vybrať polohu, aby sme mali čo najväčšiu šancu na prežitie?*

Aká poloha bude pre nás najvýhodnejšia? Určite to bude tá, kde najbližší zo strelcov (ten nás najviac ohrozuje) bude čo možno najďalej. Potrebujeme to sformulovať matematicky.

Nech sme v nejakom bode x, y štvoruholníka $\mathcal{S}$ s vrcholmi A, B, C, D a nech vzdialenosti, deliace nás od jednotlivých strelcov sú $d_1(x,y)$, $d_2(x,y)$, $d_3(x,y)$, $d_4(x,y)$. Hľadáme maximum funkcie

$$d(x,y) = \min_{(x,y)\in\mathcal{S}} \{d_1(x,y), d_2(x,y), d_3(x,y), d_4(x,y)\}. \qquad (4.2)$$

Ak bude veľkosť $d(x,y)$ väčsia, ako je dostrel (označme ho s) každého zo strelcov, sme zachránení. Inak nás skôr-neskôr dostanú aj napriek našej (relatívne) výhodnej polohe.

Budeme potrebovať vzdialenosti ľubovoľného bodu $(x_0, y_0) \in \mathcal{S}$ od strán štvoruholníka, napr. od strany AB. Rovnica priamky, danej dvomi bodmi $A = (x_A, y_A)$, $B = (x_B, y_B)$ je

$$-kx + y + (kx_A - y_A) = 0, \quad \text{kde } k = \frac{y_B - y_A}{x_B - x_A}$$

a podľa známeho vzorca je tá vzdialenosť rovná

$$d_1 = \frac{|-kx_0 + y_0 + (kx_A - y_A)|}{\sqrt{1+k^2}}.$$

pre usporiadanú dvojicu, určujúcu rozmer matice, v našom prípade to bude matica typu 2×4). Keď však zavoláme `zeros((2,4),'d')` tak sa vytvorí matica s nulami reálnymi. Pre počítač je to podstatný rozdiel, lebo reálne čísla ukladá do pamäte a pracuje s nimi úplne inak ako s celými číslami.

Uvedieme písmenové označenia pre niektoré často používané typy prvkov číselných polí: `'?'` – *booleovské hodnoty* (v Pythone sa značia identifikátormi `True`, `False`), `'b'` – *byte*, `'l'` – *celé čísla*, `'d'` – *reálne čísla*, `'D'` – *komplexné čísla*. Takže napr. `ones((3,5),'?')` bude matica 3×5, ktorej všetky prvky budú mať hodnotu `True`.

Z vyššieuvedeného príkladu (pre maticu `B`) vidíte, že operácia umocňovania sa značí dvomi hviezdičkami (to je prevzaté z FORTRAN-u). Ale tiež vidíte, že aj také veľké číslo ako 2^{80} sa zobrazí správne. Je to preto, že **Python pracuje s celými číslami s ľubovoľnou presnosťou**, čo neplatí napr. pre PASCAL či C/C++. Takže v Pylabe môžeme rátať napr. faktoriály veľkých čísel alebo binomické koeficienty (môžete si vyskúšať, sú to funkcie `factorial`, `comb`).

Na matici `D=4*ones((4,2))` vidíte, že môžeme robiť aritmetické operácie medzi maticami a číslami. Skúste si napr. `C+2` a dostanete maticu, kde ku každému prvku matice `C` je pripočítaná dvojka. Podobne funguje aj odčítanie a delenie nenulovým reálnym či celým číslom.

Takisto, teda ako operácie po prvkoch, fungujú aj aritmetické operácie medzi maticami (obyčajne rovnakého rozmeru). Vyskúšajte `A*A`, `C+C`, `F/4.0`. Takže, na rozdiel od MATLAB-u, hviezdičkový operátor nie je maticové násobenie, ako ho poznáme z algebry. Ak chceme násobiť matice či vektory takto „matematicky“, použijeme funkciu `dot`, napr. `dot(C,Bs)` urobí súčin matice `C` a stĺpcového vektora `Bs` v tomto poradí. Keby sme chceli urobiť súčin `dot(Bs,C)`, systém by nám vynadal, nesedia mu rozmery.

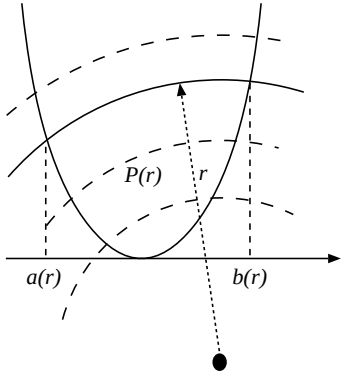
Môžeme si položiť otázku, ako zistíme rozmery viacrozmerného číselného poľa? A vôbec, aké ďalšie funkcie máme pre prácu s poľami k dispozícii? Tu prichádzame do oblasti, kde je náš pracovný jazyk – *Python* pre nás veľkou posilou, pretože je objektovo orientovaný. **Všetko, čo vytvoríme, je objekt.** Podľa toho, ako vznikol náš objekt `P`, má vždy nejaký dátový typ. Ten zistíme príkazom `type(P)`. Ako to poznáte z iných objektovo orientovaných jazykov, každý objekt má tiež svoje **dáta a metódy** (keď použijeme terminológiu z C++). K nim sa dostaneme v Pylabe cez bodkové označenie.

Napr. ak `A` je číselné pole (to je u nás dátový typ `ndarray`), potom jeho metódy (funkcie) sú `A.max`, `A.sum`, `A.reshape`, `A.transpose` a veľa iných. Jeho dáta (atribúty) sú napr. `A.shape`, `A.dtypechar`, `A.ndim`. No odkiaľ sme sa to dozvedeli? To si musíme pamätať, či čo? Žiadny strach,

4.2 Numerické integrovanie

Príklad 4.3 (Príklad o koze). *V bode* $(1,-3)$ *(vzdialenosti budeme merať v metroch) je priviazaná koza. V parabole* $y = x^2$ *je vrbina, ktoré tá koza používa ako potravu. Mimo tej paraboly je púšť, kde sa nedá nájsť žiadna obživa. Ako dlhý má byť špagát, na ktorom je koza priviazaná, aby obhrýzla presne svoju dennú dávku* $10\ m^2$ *vŕb?*

Situáciu ukazuje obrázok:



Z rovnice hraničnej kružnice (kam až dosiahne povraz) dostávame

$$(x-1)^2 + (y+3)^2 = r^2 \;\Rightarrow y_k = \sqrt{r^2-(x-1)^2} - 3.$$

Keď sa dohodneme na označení podľa obrázku, teda $P(r)$ bude plocha spásateľnej oblasti, potom potrebujeme určiť takú hodnotu r, aby

$$P(r) = \int_{a(r)}^{b(r)} \left(y_k - x^2\right) \mathrm{d}x = \int_{a(r)}^{b(r)} \left(\sqrt{r^2-(x-1)^2} - 3 - x^2\right) \mathrm{d}x = 10.$$

Priesečníky $a(r), b(r)$ paraboly a kružnice sú reálne korene rovnice

$$x^2 = \sqrt{r^2-(x-1)^2} - 3$$

. Umocnením ju ľahko upravíme na polynomickú

$$x^4 + 7x^2 - 2x + 10 - r^2 = 0 \tag{4.1}$$

Dôležité je uvedomiť si, že pre ľubovoľné rozumné r vieme vypočítať hodnoty funkcie $P(r)$, ktorej koreň hľadáme. Najskôr určíme reálne korene $a(r)$, $b(r)$ polynomickej rovnice (pomocou funkcie `roots`) a potom

že na jednom riadku môžeme zadať viac príkazov, ak ich oddelíme bodkočiarkou (platí aj v Pythone nielen v Pylabe). Na spájanie matíc (vedľa seba alebo pod sebou) máme ešte funkciu `concatenate`.

Cvičenie 1.1 *Vytvorte „šachovnicu", teda maticu* 8×8*, kde na bielych poliach budú nuly a na čiernych jedničky. Myslí sa inteligentný algoritmus, s čo najmenšou námahou a nie otrocké zadávanie všetkých prvkov. A samozrejme taký aby sa dal ľahko zovšeobecniť aj na väčšie matice.*

V numerike aj v úlohách inžinierskej praxe sa stretávame s pásovými maticami. Príkaz `diag` nám umožňuje poskladať maticu zo šikmých „pásov", rovnobežných s hlavnou diagonálou. Skúste takto vytvoriť maticu `M` rozmeru 10×10, ktorá bude mať na hlavnej diagonále trojky, nad ňou čísla -1 a pod diagonálou čísla -2; ostatné prvky matice `M` sú nulové.

Často potrebujeme vektory, ktorých prvky sú „pravidelne rozmiestnené" medzi hodnotami `v_beg` *(počiatočná hodnota)* a `v_end` *(koncová hodnota)*. Ak poznáme počet prvkov `n` takého vektora `v`, vytvoríme ho jednoducho príkazom

$$\texttt{v} = \texttt{linspace}(\texttt{v\_beg}, \texttt{v\_end}, \texttt{n}).$$

Vektory ekvidištančných (rovnako od seba vzdialených) hodnôt, generované pomocou `linspace` využívame často na kreslenie grafov (kriviek, plôch). Napríklad príkazmi

$$\texttt{x} = \texttt{linspace}(-\texttt{pi}, 3*\texttt{pi}, 100); \quad \texttt{y} = \texttt{sin}(\texttt{x}); \quad \texttt{plot}(\texttt{x}, \texttt{y})$$

vytvoríme 100-prvkový vektor `x` hodnôt, rovnomerne pokrývajúcich interval $\langle -\pi, 3\pi \rangle$, ďalej vypočítame hodnoty sínusu vo **všetkých** týchto bodoch `x` a vykreslíme graf funkcie $y = \sin x$ pre x v intervale $\langle -\pi, 3\pi \rangle$. Skúste si to, nech vidíte už aj nejaký obrázok :-). Neskôr si o grafike v Pylabe povieme podrobnejšie.

Na vytvorenie vektora celočíselných ekvidištančných hodnôt je výhodnejší príkaz `arange(i_beg, i_end, step)`, ktorý vytvorí vektor ekvidištančných hodnôt s krokom `step` začínajúci hodnotou `i_beg`, pričom koncová hodnota nie je väčšia ako `i_end - 1`. Napríklad príkaz `arange(1, 10, 2)` vytvorí vektor `(1, 3, 5, 7, 9)`, ale `arange(1, 9, 2)` dá vektor `(1, 3, 5, 7)`. Ak nezadáme argument `step`, berie sa krok 1 (vektor za sebou idúcich celých čísel), napr. `arange(1, 6)` je vektor `(1, 2, 3, 4, 5)`. Ak zadáme jedinú hodnotu, napr. `arange(n)`, dostaneme vektor n hodnôt $0, 1, \ldots, n-1$. Pythonský príkaz `range` sa správa takisto, ale namiesto číselného vektora vracia zoznam.

Ak si cez menu obrázkového okna urobíte vhodný výrez, môže váš obrázok vyzerať podobne ako vidíte tu (čiarkovanú čiaru – pre vrstevnicu prvej funkcie nebolo až tak ľahko urobiť, ale už by ste to mali dokázať aj vy).

Z obrázku jednoznačne vidieť, že riešenia sústavy v prvom kvadrante sú štyri. Ak si zväčšíte obrázkové okno do ich tesnej blízkosti, môžete odčítať aj ich približné súradnice:

$$(0.38, 1.06),\ (1.23, 0.55),\ (1.02, 1.54),\ (1.58, 1.36).$$

Na riešenie nelineárnych rovníc a ich sústav máme v Pylabe (takisto ako v MATLAB-e) funkciu `fsolve`, je schovaná v submodule `scipy.optimize`. Prvým argumentom tejto funkcie je funkcia, ktorej keď zadáme (vektorový) argument – v našom prípade (x, y), vráti nám vektor hodnôt $(f_1(x, y), f_2(x, y))$. Napíšme ju v editore a uložme do súboru `nltrig.py`:

```
from numpy import sin, cos

def nlfct(xy):
    x,y=xy   # xy je dvojzlozkovy vektor, roztrhneme ho
    f1=sin(x*y*y)-cos(x*x-y)+0.2
    f2=x**3+y**3-3*x*y
    return (f1,f2)
```

Druhým argumentom pre `fsolve` je počiatočné priblíženie riešenia, teda napr. tie hodnoty, čo sme odčítali z vrstevnicového grafu. Takže načítame našu funkciu do interaktívneho prostredia cez `run nltrig.py` a skúsime

```
from scipy.optimize import fsolve
r=fsolve(nlfct,(1.23, 0.55)) # bude r=(1.2328735, 0.5521787)
```

Ak si dáte príkaz `nlfct(r)`, ktorý vypočíta hodnoty funkcií $f_1(r)$, $f_2(r)$ v riešení, uvidíte, že sú veľmi blízke nule, ako by to aj malo byť. Nám vyšli okolo `(-4.552e-15, -3.997e-14)`. Ostávajúce tri riešenia si urobte samostatne. Zaujímavé, že pre tretie a štvrté riešenie vyšli hodnoty $f_1(r)$, $f_2(r)$ horšie, rádovo okolo 10^{-12}. Dá sa to tiež názorne vysvetliť, ak si znázorníte pre obidve funkcie susedné „blízke“ vrstevnice, napr. pre hladiny -0.1, 0.1. V okolí tých dvoch posledných riešení sú vrstevnice hustejšie, preto sa funkčné hodnoty rýchlejšie menia, ak sa mierne vzdiaľujeme od riešenia. Ale neverte nám, skúste si to sami.

Príklad 4.2 *Zistite, koľko koreňov má rovnica* $e^x = ax^2 - 1$ *v závislosti od reálneho parametra* a.

vynulujeme celú príslušnú submaticu matice `A`. To je jeden zo spôsobov, ako sa vyhnúť explicitným cyklom (typu `for`, `while`) a zrýchliť beh programu.

V algebre ste často robili operácie s riadkami matice, napr. nejaký násobok prvého riadku ste pričítali k druhému riadku matice `A` a výsledok zapísali do druhého riadku. To sa urobí aj v Pylabe ľahko (nezabúdajme na indexovanie riadkov a stĺpcov od nuly):

$$\texttt{A}[1] = \texttt{A}[1] + 3.5 * \texttt{A}[0]$$

Ďalší príklad:

$$\texttt{dx} = \texttt{x}[1:] - \texttt{x}[:-1]$$

vypočíta vektor diferencií $\Delta x_i = x_{i+1} - x_i$ pre $i = 1, 2, \ldots, n-1$, ak `x` je n-prvkový vektor. Záporné indexy sú špecialitou Pythonu a počítajú sa od konca, takže `x[:-1]` je vektor `x` bez posledného prvku.

Pomocou dvojbodkoveho označenia by ste si mohli skúsiť urobiť tú šachovnicovú maticu núl a jedničiek, ktorú sme hore konštruovali z blokov. Stačilo by vyrobiť vektory prvých dvoch riadkov a tie potom „ob dva“ opakovať.

Prechádzame k operátorom porovnávania (t. j. `<`, `>`, `<=`, `>=`, `==`, `!=`). Predpokladajme, že `A` je náhodná matica, vytvorená ako `A=rand(4,4)`. Skúste si, čo urobia príkazy `A>0.7`, `A<0`, `A != A`, `A == A`. Ako vidíte, výsledok je booleovská matica s rovnakým rozmerom ako matica `A`, v ktorej hodnoty `True` sú na miestach, kde je podmienka porovnávania splnená a hodnoty `False` tam, kde nie je.

Operátory porovnávania môžeme tiež kombinovať s logickými operátormi `&`, `|`, `~`. Pre tých, čo poznajú Python – nepoužívajte na číselné polia operátory `and`, `or`, `not`, lebo tie vám vynadajú (nevedia porovnávať polia „po prvkoch“).

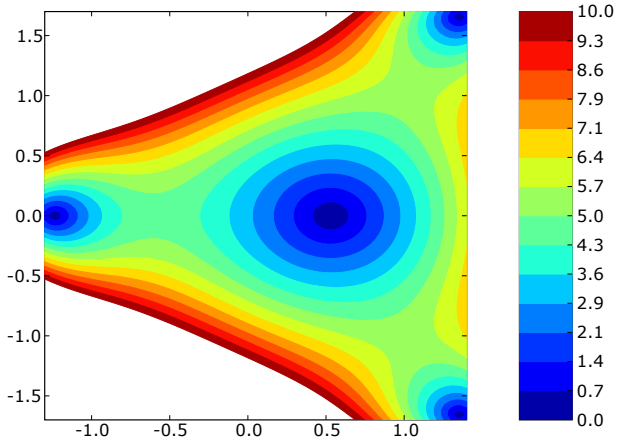
Najkrajšie na tom všetkom je, že výsledok porovnávania (booleovská matica) sa dá použiť na ďalšiu manipuláciu s tými prvkami matice, kde je pravdivostná hodnota `True`. Napr.

$$\texttt{A}[(\texttt{A} > 0.8) \mid (\texttt{A} < 0.1)] = 0$$

vynuluje v matici `A` všetky prvky, ktoré sú väčšie ako 0.8 alebo menšie ako 0.1. Porozmýšľajte, ako by ste vynulovali všetky prvky, ktorých absolútna hodnota je menšia ako dané kladné číslo ε, napr. $\varepsilon =$ `4.4e-16`.

Pekný príklad: zistime počet prvkov v náhodnej matici 100×100, ktoré sú väčšie ako 0.7 (keďže viete niečo z pravdepodobnosti, mali by ste tušiť, koľko ich bude):

```
clevels=linspace(0,10,15)   # vrstevnicove hladiny
contourf(X,Y,Fxy,clevels)   # vyplneny vrstev. graf
colorbar()                  # pridanie fareb. stupnice
```



Ako ľahko zistíte príkazom `Fxy.max()` maximálna hodnota $f(x, y)$ na zvolenej sieťke je asi 48.23. My sme znázornili 15 vrstevnicových hladín od 0 do 10, preto je časť obrázku biela. Jasne na ňom vidieť polohu koreňov, pretože okolo nich sú uzavreté, skoro eliptické vrstevnice (dva korene sú v dolnom a hornom pravom rohu a dva na reálnej osi).


```
cq=[1, 5, -6]          # polynom q
cs=polymul(cp,cq)      # vysledok nasobenia - polynom
cpd,zv=polydiv(cs,cq) # s/q = p (cpd = cp plati?)
```

Maticami sa zaoberáme dosť aj preto, aby sme vedeli riešiť sústavy lineárnych rovníc. Ak matica systému je štvorcová a regulárna, t. j. s nenulovým determinantom, môžeme použiť funkciu `solve`. Napr:

```
A=rand(1000,1000)    # nahodna matica 1000 x 1000
b=rand(1000)         # nahodna prava strana
x=solve(A,b)         # x je riesenie syst. A x = b
norm(dot(A,x)-b)     # velkost vektora A x - b
                     # musi byt velmi male cislo
```

Pravá strana b systému môže byť aj matica typu $n \times m$ ak matica systému je typu $n \times n$. Je to akoby m pravých strán naraz (stĺpce matice b). Napr.

$$\texttt{Ai} = \texttt{solve}(\texttt{A}, \texttt{eye}(1000, \texttt{dtype} = \texttt{'d'}))$$

vypočíta inverznú maticu k matici `A`. Ale nikdy to tak nerobte, je to neefektívne. A vôbec, inverznú maticu potrebujeme skutočne zriedkavo. Potrebujeme riešiť sústavy lineárnych rovníc. To vieme pomocou metód numerickej analýzy aj pre sústavy so všeobecnou obdĺžnikovou maticou (KAUKIČ, 1998; BUŠA, 2006).

V Pylabe by to mohlo vyzerať takto:

```
fi=linspace(0,2*pi,120); x=2*cos(fi); y=sin(fi)
ro=sqrt(x**2+y**2)
polar(fi,ro)
```

Ak si pozriete obrázok, iste uznáte, že naša elipsa si zaslúži prívlastok „prvoaprílová“. Vysvetlite, prečo vyzerá tak nezvykle ...

3.3 Znázorňovanie funkcií dvoch premenných

Príklad 3.8 *Nakreslime vrstevnicový graf funkcie dvoch premenných*

$$z = \sin(xy^2) - \cos(x^2 + y) \text{ pre } -2 \leqq x \leqq 2,\ -2.5 \leqq y \leqq 2.5.$$

Najskôr vygenerujeme ekvidištančné vektory `xx`, `yy` hodnôt na súradnicových osiach v zadaných intervaloch:

```
xx=linspace(-2,2,80);     yy=linspace(-2.5,2.5,100)
```

a tie potom použijeme v príkaze `meshgrid`, ktorý vráti matice `X`, `Y` x-ových a y-ových bodov obdĺžnikovej sieťky, na ktorej sa musia počítať funkčné hodnoty z. Napr. pre `xm=[1,2,3]`, `ym=[0,1,2,3]` vráti `meshgrid(xm,ym)` tieto matice:

```
        1  2  3              0  0  0
Xm =    1  2  3       Ym =   1  1  1
        1  2  3              2  2  2
        1  2  3              3  3  3
```

V našom prípade budú mať matice `X`, `Y` rozmery 80×100 a dostaneme ich príkazom

```
X,Y=meshgrid(xx,yy)
```

Použijeme ich na výpočet funkčných hodnôt a nakreslenie vrstevnicového grafu (contour plot) príkazom `contour`:

```
Z=sin(X*Y*Y)-cos(X*X-Y)
contour(X,Y,Z,20)
```

Dostanete nasledujúci obrázok. Ako sa dá dočítať z nápovedy k príkazu `contour`, jeho najjednoduchšie volanie je `contour(Z)`. Rozmyslite si, čo

niec priradenie hodnôt prvkom matice (naraz dvom – pretože Hilbertova matica je symetrická, t. j. $h_{ij} = h_{ji}$). Na tomto príklade dobre vidieť použitie indentácie. Cyklus `for j in range(i,n):` je vnorený do prvého cyklu, pretože je na riadku odsadený vzhľadom na ten prvý cyklus. Príkaz priradenia `H[i,j]=H[j,i]=1.0/(i+j+1)` je súčasťou oboch cyklov, ale príkaz `return` už ani do jedného z cyklov nepatrí, pretože začína na rovnakej úrovni indentácie ako vonkajší cyklus. Keby sme pridali do súboru ďalší riadok, ktorý by začínal príkazmi na úrovni `def`, tak tieto príkazy by už nepatrili do funkcie `hilb`. Na takúto logiku sa dá pomerne rýchlo zvyknúť. No a obsah súboru bude nasledujúci:

```
from numpy import empty

def hilb(n):
    H=empty((n,n),'d')          #inicializacia matice H
    for i in range(n):          # priradenie prvkov
        for j in range(i,n):
            H[i,j]=H[j,i]=1.0/(i+j+1)
    return H
```

Keď ten súbor uložíte (pre `nedit` je to klávesová skratka `Ctrl-S`), nechajte si editor otvorený (väčšinou programy nebývajú bez chýb a treba ich ladiť, teda opakovane prepisovať v editore). Vráťte sa do interaktívneho prostredia Pylabu a príkazom `run hilbert.py` načítate do Pylabu všetko, čo je v tom súbore definované. V našom prípade je to len funkcia `hilb` a môžete sa pomocou príkazov `who`, `whos` presvedčiť, že ju máte.

Naša funkcia sa vyvolá (použije) podobne ako vstavané funkcie, teda, že zadáme meno funkcie a nejaké (konkrétne) argumenty (alebo, keby argumenty neboli, tak len prázdne zátvorky). Napr. `hilb(5)`, `hilb(12)`. No nie je blbuvzdorná, lebo napr. vykoná sa aj `hilb(5.735)`. Ale vždy začíname tak, že nám ide o správnu funkcionalitu programu a predpokladáme inteligentného užívateľa (obyčajne prvý, kto to testuje, je sám tvorca programu :-) a až neskôr sa vrátime k ošetreniu chýb vstupu a iným menej podstatným veciam.

Zapamätajme si, že **dvojbodky na konci riadkov sú súčasťou syntaxe** a píšu sa len v týchto prípadoch:

1. za definíciou funkcie, napr. `def factorial(m):`,
2. na začiatku cyklov `for`, `while`,
3. v podmienkach, za kľúčovými slovami `if`, `elif`, `else`, `finally` a tiež pri ošetrovaní výnimiek (kľúčové slová `try`, `except`).

V tomto listingu sme vynechali príkazy pre tretí a štvrtý obrázok, lebo je to všetko na jedno kopyto.

V obrázku, ktorý vidíte, sme manuálne upravili rozmery obrázkov (pomocou `Subplot Configuration Tool` voľby v menu obrázkového okna), lebo inak by popisy osí a nadpisy grafov boli príliš natesno. Dá sa to, samozrejme, aj priamo cez atribúty aktuálneho obrázka, teda napr.:

```
cf=gcf()          # get current figure
cf.subplotpars.left=0.06; cf.subplotpars.right=0.94
```

Že sa to tak volá, zistili sme doplňovaním pomocou `TAB`-u. Súradnice sa udávajú normalizované, od 0 do 1. Všimnite si tiež matematické výrazy v textoch (sú to tie, ohraničené dolármi) – syntax, čo tam vidíte, je prevzatá z LaTeX-u.

Príklad 3.6 *Znázornite v tom istom grafickom okne funkcie*

$$f_1(x) = \frac{2}{1+x^2} - 1, \quad f_2(x) = \sin\left(x^3\right)$$

a kružnicu $x^2 + y^2 = 0.81$, *pričom body kružnice budú znázornené krúžkami po každých* 10°.

Najskôr (keďže to nemáme zadané) sa rozhodneme pre interval na osi x, na ktorom chceme funkcie znázorňovať. Napr. teraz si zvolíme $-2 \leqq \leqq x \leqq 2$.

Zostrojíme dostatočne „hustý“ vektor bodov, rovnomerne pokrývajúci daný interval (pri kreslení grafu sa jednotlivé body spoja úsečkami; pre hladké zobrazenie na obrazovke obyčajne stačí rádove stovka bodov). Potom vypočítame hodnoty funkcií f_1, f_2 v týchto bodoch.

```
x=linspace(-2,2,100)
f1=2.0/(1+x*x)-1; f2=sin(x**3)
```

Tieto dve funkcie vykreslíme, pričom predpíšeme, aby druhá funkcia bola nakreslená čiarkovane a zelenou čiarou (*green*)

```
plot(x,f1,x,f2,'--g')
```

Na zostrojenie kružnice môžeme použiť parametrický tvar jej rovnice

$$x = r\cos\varphi, \quad y = r\sin\varphi, \quad \varphi \in \langle 0, 2\pi\rangle$$

. Zostrojíme vektor uhlov `fi`, body `xt`, `yt` na kružnici a dokreslíme tú kružnicu červenými kolieskami:

doplní. Teda príkaz na import funkcie `solve` bude:

```
from scipy.linalg import solve.
```

Modul na kreslenie, `pylab`, používame obyčajne tak, že z neho importujeme všetko, teda `from pylab import *`. Zvyknite si, že **všetky importy dávame na začiatok zdrojového súboru**, nikdy nie roztratené sem-tam alebo niekde v našich definovaných funkciách. Inak sa budete veľmi čudovať, prečo raz máte a raz nemáte niečo k dispozícii. Ladenie programu s roztratenými importami by bolo vrcholne nepríjemné.

Treba si ujasniť ešte jednu fundamentálnu vec: **ak je nejaký objekt dostupný v interaktívnom prostredí, nemusí byť dostupný vo vašom zdrojovom súbore**. Každý Pylabský zdrojový súbor sa chová ako „minimodul“ a má svoj vlastný priestor mien. Napríklad po spustení Pylabu máte v interaktívnom prostredí k dispozícii funkcie `array`, `dot`, `solve`, `zeros`, `bmat`, `empty` a mnoho ďalších. Je to zaistené tým, že sme ich pridali do vhodných inicializačných súborov, ktoré si Pylab načíta pri štarte. Ale to váš zdroják nerobí, a preto všetky tieto funkcie musíte importovať, ak ich chcete používať. No a samozrejme, načítať vami definované funkcie vykonaním vášho zdrojáku cez príkaz `run`, napr. `run factorial.py`.

Príklad 2.2 *Napíšme funkciu na výpočet faktoriálu nezáporného celého čísla n (uložte ju do súboru* `factorial.py`*).*

```
def factor(n):
    assert(type(n)==int and n>=0), "Need natural number"

    fct=1
    if n in [0,1]:
        return fct
    else:
        for k in range(2,n+1):
            fct = k*fct   # alebo ako v C-cku: fct *=
    return fct
```

Príklad 2.3 *Vytvoríme funkciu na výpočet n-tého člena a_n Fibonacciho postupnosti, ktorá je daná predpisom*

$$a_0 = 0, a_1 = 1; \quad a_{k+2} = a_{k+1} + a_k \ \textit{pre}\ k = 0, 1, \ldots.$$

```
def fib(n):
    assert(type(n)==int and n>=0), "Need natural number."
    if n in [0,1]:
        return n
```

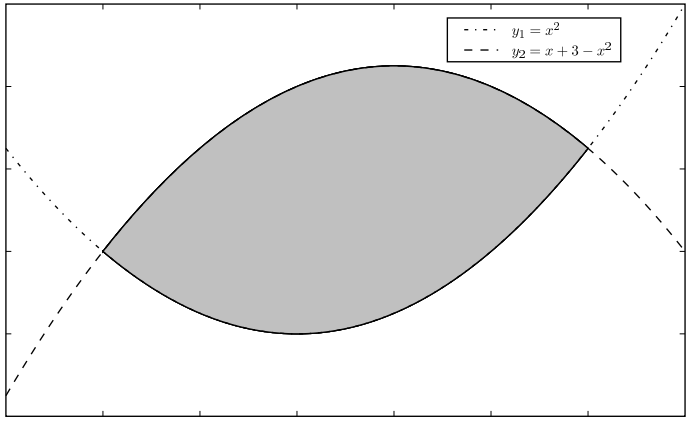
V podstate nám ide o farebné vyplnenie oblasti medzi dvomi krivkami. Pylab má funkciu `fill` na farebné vyplnenie mnohouholníka. Naše dve krivky tvoria vlastne veľký mnohouholník, ak jednu z nich budeme prechádzať v obrátenom poradí.

Priesečníky kriviek y_1, y_2 sa ľahko zistia riešením kvadratickej rovnice

$$y_1 = y_2 \;\Leftrightarrow\; 2x^2 - x - 3 = 0 \;\Leftrightarrow\; x = -1, x = 3/2.$$

```
x=linspace(-1,3/2.0,100)
yp1,yp2=x*x,x+3-x*x
xpoly=concatenate(( x,   x[::-1]))
ypoly=concatenate((yp1,yp2[::-1]))
fill(xpoly,ypoly,facecolor="#C0C0C0")
```

Ak si tieto príkazy píšete do súboru, musíte dať na jeho začiatku príkaz `from pylab import *`, inak by vám systém vypisoval, že nepozná funkcie `linspace`, `concatenate`, `fill`. Platí to aj pre nižšieuvedené ukážky grafiky, ak ich budete spúšťať zo súborov. Pripomeňme, že `x[::-1]` je vektor `x` s opačným poradím prvkov. V príkaze `fill` vidíte, že farby možno zadávať podobne, ako sa to robí v HTML-súboroch, ako reťazce RGB (Red, Green, Blue), pričom `#000000` je čierna, `#FFFFFF` biela farba a napr. `#0000AA` je tmavomodrá.



Aby to vyzeralo krajšie, potiahneme tie paraboly trochu aj mimo uvažovanej oblasti, napr. pre $x \in \langle -1.5, 2.0 \rangle$, rozlíšime ich typom čiar, pridáme legendu a nadpis grafu.


```
    Ph=0            # pocet hran
    Zh=[]           # zoznam hran
    odpad=0         # pocet zamietnutych hran
    while True:
        u,v=rndint(1,n),rndint(1,n)
        # pripustne su len hrany pre u<v
        # a take, ktore este nemame
        if u<v and (not (u,v) in Zh):
            Zh.append((u,v)); Ph += 1
        else:
            odpad += 1
        if Ph==m:
            break
    return Zh, odpad
```

Všimnite si, že pre pohodlnosť sme si pri importe premenovali funkciu `random_integers` – dali sme jej kratšie meno. Ak sa čudujete, načo nám je premenná `odpad`, tak prakticky nanič. Ale môžeme si podľa nej urobiť predstavu, aká je naša metóda hrubej sily (ne)účinná. Viac ako polovička generovaných hrán sa nepoužije. Skúste si vyvolať našu funkciu (niekoľkokrát) tak, ako to potrebujeme, teda

```
Zh,odp=randgraph(12,53)
```

a uvidíte, že v premennej `odp` bude vždy okolo 200 odvrhnutých hrán. Potom, na „produktívne“ využitie upravte funkciu `randgraph` bez premennej `odpad`, čím sa jej zdroják dosť zjednoduší.

Zaujímavosťou funkcie `randgraph` je jej správanie sa pri volaniach ako napr. `randgraph(12,67)`. Odporúčame vyskúšať. A hlavne potom upraviť tak, aby sa podobné psie kusy neopakovali (je možné, že v panike aj počítač resetnete, ale nerobte to :-). Radšej si spomeňte na niektoré elementárne fakty z kombinatoriky.

Teraz môžeme ten zoznam hrán uložiť do textového súboru s názvom `gr12_53.txt` príkazom:

```
save('gr12_53.txt',Zh,'%i')
```

Tretí, nepovinný parameter je formátovací reťazec v štýle C-čka, teda `"%i"` znamená uloženie v celočíselnom formáte (`"%d"` by bol formát pre reálne čísla v dvojitej presnosti – typ `double` v C-čku). Súbor `gr12_53.txt` si môžete pozrieť a prípadne upravovať v textovom editore; vyskúšajte, čo to urobí, ak zavoláte funkciu `save` bez tretieho parametra.

Teraz zničíme premennú `Zh`, ako by sme to urobili s hocijakým Pylabským objektom, teda príkazom `del(Zh)`. Môžete sa presvedčiť cez príkaz

Môžete si tie dva obrázky porovnať. Iste uznáte, že ten druhý vyzerá lepšie. Len je škoda, že pri čiernobielom vytlačení všetky tie tri čiary vyzerajú skoro rovnako. To sa budeme teraz snažiť napraviť.

Keď máme tie naše tri čiary v zozname `lines`, roztriedime ich do troch premenných príkazom `lsin,lt5,lt9=lines`. Keď si zas známym spôsobom pozrieme metódy pre nastavenia (napr. napíšeme `lt5.set` a stlačíme kláves TAB), nájdeme metódu `set_linestyle` (alebo skrátený názov `set_ls`) a z nápovedy sa dozvieme, aké rôzne štýly čiar môžeme nastaviť. Urobíme

```
lsin.set_ls('-'); lt5.set_ls('-.'); lt9.set_ls(':')
```

a budeme mať krivku pre sínus znázornenú plnou čiarou, Taylorov polynóm piateho stupňa bodkočiarkovane a polynóm deviateho stupňa bodkovane.

V nasledujúcej tabuľke vidíme rôzne možné štýly. Prvé štyri sú pre čiary, všetky ďalšie znázorňujú zadané body (x, y) značkami.

Tabuľka 1: Štýly čiar v Pylabe

`-`	neprerušovaná čiara		`--`	čiarkovaná čiara
`-.`	bodkočiarkovaná čiara		`:`	bodkovaná čiara
`.`	body		`,`	obrazovkové pixely
`o`	kruhy		`^`	trojuholníky základňou dole
`v`	trojuholníky obrátene		$<$	trojuholníky doprava
$>$	trojuholníky doľava		`s`	štvorce
`+`	symboly plus		`x`	symboly tvaru x
`D`	kosoštvorce		`d`	tenké kosoštvorce
`1` až `4`	trojnožky rôzne otočené		`h`	šesťuholníky
`H`	otočené šesťuholníky		`p`	päťuholníky

Podobne vieme nastaviť farbu čiar a ich hrúbku. Napr. všetky tri čiary nakreslíme čiernou farbou:

```
lsin.set_c('k'); lt5.set_c('k'); lt9.set_c('k').
```

Štýly a farby čiar sa dajú nastavovať priamo v príkaze `plot`, o čom si prečítajte podrobnejšie v nápovede k tomuto príkazu. Takže, ak našu grafiku vymažeme príkazom `clf()` (clear figure) a dáme

```
plot(xx,sin(xx),'-k',xx,T5,'-.k',xx,T10,':k')
```

dostaneme tiež čierne čiary rôznych štýlov.

príkazom `close`, ako sme to urobili povyššie. Inak môže byť neúplný (autor hovorí z vlastnej skúsenosti). Časť dát môže ešte čakať na zápis v pamäťovom bufferi – ten môžeme vyprázniť aj volaním metódy `flush`, teda v našom prípade by to bolo `outfile.flush()`.

Dáta zo vzniknutého binárneho súboru dostaneme použitím funkcie `fromfile` a priradíme ich do novej premennej `Rs`:

```
Rs=fromfile("Rbig.bin",int)
```

Druhý parameter hovorí, že typ prvkov poľa je celočíselný (iné možné typy sú napr. `float`, `complex`, `bool`, `str`). Pole, ktoré dostaneme ako výstup z funkcie `fromfile` je vždy jednorozmerné, ale vieme jeho rozmery ľahko upravovať priradením do atribútu `shape`, teda z vektora `Rs` o milión prvkoch dostaneme maticu typu 1000×1000 jednoducho príkazom

```
Rs.shape=(1000,1000)  # tvar Rs sa zmeni na 1000 x 1000
```

Sme zvedaví, či pôvodná matica `R` a načítaná matica `Rs` sú rovnaké. Presvedčime sa o tom príkazom

```
all(Rs==R) # Vsetky prvky Rs su rovne zodpovedajucim prvkom R?
```

Ak nám tento príkaz vypíše `True`, obe matice sú rovnaké. Dúfame, že je to aj u vás tak.

Ako je to s časom čítania textového a binárneho súboru? Aby sme to zistili, importujeme si z modulu `time` funkciu `time`, ktorá meria čas aj na mikrosekundy. Potom si zapíšeme čas pred začiatkom výpočtu. Po jeho skončení vypíšeme na obrazovku rozdiel medzi tým počiatočným a aktuálnym časom:

```
from time import time
tb=time(); Rs=fromfile("Rbig.bin",int);time()-tb #0.0127 sec.
tt=time(); load("Rbig.txt",Rt);        time()-tt #3.345  sec.
```

V komentároch vidíte aktuálne časy, ktoré sme dostali na našom katedrovom serveri (Athlon64 3500+, RAM 2GB). Zasa je čítanie z binárneho súboru nepomerne rýchlejšie (asi 270-krát v tomto prípade).

Pripomeňme si, že Taylorov rozvoj funkcie $y = \sin x$ je

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots + (-1)^k \frac{x^{2k+1}}{(2k+1)!} + \cdots$$

. Napíšte funkciu, ktorá vráti hodnoty Taylorovho polynómu stupňa n v zadaných bodoch x, uložte si ju ako súbor `taylsin.py`:

```
from numpy import array

def taylsin(x,n):
    assert(type(n)==int and n>0)
    x=array(x,'d')   # ak bol zoznam, urobime pole
    x2=x*x
    f=1.0            # faktorial - inicializacia
    T=x.copy()       # Taylorov polynom - inicializacia
    xa=x.copy()      # aktualna mocnina x v Tayl. pol.
    for k in range(3,n,2):
        f  *= -((k-1)*k)  # - pre striedanie znamienok
        xa *= x2          # exponent x sa zvacsi o 2
        T += xa/f         # dalsi scitanec do T
    return T
```

Všimnime si niektoré pozoruhodnosti v tejto funkcii. Použitím príkazu `x=array(x,'d')` zaistíme, že ďalej vo funkcii bude premenná `x` fungovať ako číselné pole (aj z jedného čísla sa dá vyrobiť pole). Bez toho by sme nemohli používať aritmetické operácie, napr. `x*x`. Používali sme C-čkovský variant zápisu priradení, teda `T += xa/f` namiesto `T = T + xa/f`.

V príkazoch na inicializáciu `T`, `xa` sme zobrali **kópie** poľa `x`, pretože napr. `T=x` by spôsobilo prepisovanie poľa `x`; priradenie pre polia nevytvára novú kópiu, ale len ďalší odkaz (referenciu) na pôvodné pole. Obidve premenné budú ukazovať na to isté miesto v pamäti.

Teraz už len ostáva zvoliť vhodný interval na kreslenie (stačí od nuly, lebo aj sínus aj tie Taylorove polynómy sú stredovo symetrické podľa počiatku súradníc) a použiť príkaz `plot`

```
xx=linspace(0,4,120)
T5,T10=taylsin(xx,5),taylsin(xx,10)
lines=plot(xx,sin(xx),xx,T5,xx,T10)
```

Už predtým ste si možno všimli, že kedykoľvek ste do obrázka niečo pridali (príkazmi `plot`, `title`, `text`, `xlabel`, ...) tak sa na obrazovku niečo vypísalo, teda tie príkazy vracajú nejaké objekty. Sú to objekty toho typu,

Vidíte, že na nakreslenie jednoduchého grafu nám stačí niekoľko príkazov. Aktuálne okno s grafikou (`current figure` v terminológii Pylabu) môžete modifikovať veľa spôsobmi, napr. skúste si:

```
grid(1)              # kresli sietku na grafe
title("Graf funkcie sin")  # titulok grafu
xlabel("Os x")             # popis osi x
ylabel("Os y")             # popis osi y
```

Zasa treba zdôrazniť, že hoci v interaktívnom prostredí Pylabu máte všetky príkazy pre grafiku (napr. `plot`, `grid`, `title`, `legend`) k dispozícii, vo vašich zdrojákoch ich musíte importovať pomocou príkazu „importuj všetko z modulu pylab“, t. j.:

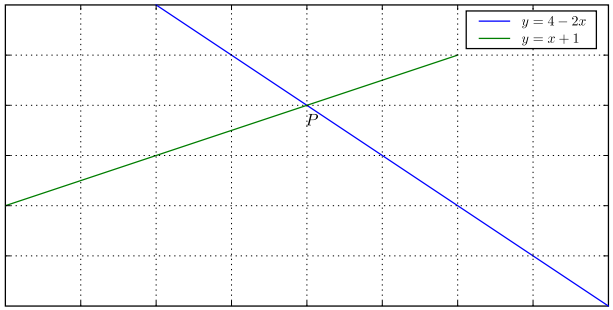
```
from pylab import *
```

Príklad 3.1 *Znázornime riešenie sústavy rovníc*

$$\begin{aligned} 2x + y &= 4, \\ -x + y &= 1. \end{aligned}$$

Riešením je bod P, ktorý je priesečníkom priamok $y_1 = 4 - 2x$, $y_2 = x + 1$. Znázornime ich časti ako úsečky, určené bodmi $A_1 = (3, -2)$, $B_1 = (0, 4)$ resp. $A_2 = (-1, 0)$, $B_2 = (2, 3)$.

```
# do plot-u osobitne zadavame
# x-ove a y-ove suradnice bodov
plot([3,0],[-2,4],[-1,2],[0,3])  # dve usecky
grid(True)
legend(('y=4-2x','y=x+1'))
t=text(1.05,1.7,'P')
title("Solution of system of linear equations")
```



Vyzerá to celkom pekne, len sa zdá, že to písmeno *P* by mohlo byť trochu inde (polohu sme vybrali pomocou súradníc, ktoré Pylab ukazuje, keď sa pohybujete kurzorom myši v obrázku). Nie je problém to dodatočne napraviť. Sme v Pythone a všetko je objekt. Aj náš obrázok je poskladaný z objektov.

Koncepcia grafiky v Pylabe je založená na tom, že máme obrázkové okná (`figures`) a v každom z nich môže byť niekoľko súradnicových systémov, ktoré môžeme umiestňovať v obrázku, kam sa nám zachce, aj jeden cez druhý. Grafické objekty (úsečky, body, mnohouholníky, atď.) sa pridávajú vždy do aktuálnych súradnicových osí.

Aktuálny obrázok a aktuálne súradnicové osi (ak sme nezatvorili všetky grafické okná) získame príkazmi:

```
fig=gcf()       # get current figure
axs=gca()       # get current axes
```

Keď si chceme nám už dôverne známym spôsobom (napísať `axs.` a stlačiť `TAB`) pozrieť metódy a dáta objektu `axs`, vychŕli to na nás okolo 230 možností. Namiesto písmena *P* by mohol byť ľubovoľný text, skúsime teda doplniť pomocou klávesu `TAB` text `axs.te`. Zistíme, že také objekty v aktuálnych osiach máme dva: `text`, `texts`. Ten prvý je metóda na pridávanie textu do obrázka na zadanej pozícii, ako zistíme z helpu (teda cez `axs.text?` alebo `?axs.text`). Druhý objekt, `texts`, je zoznam, lebo keď ho chceme vypísať cez `axs.texts`, dostaneme niečo ako

```
[<matplotlib.text.Text instance at 0x2aaab6c2f248>].
```

Ten zoznam má jediný prvok, inštanciu objektu `matplotlib.text.Text`. Teda náš text dostaneme ako nultý prvok zoznamu, t. j. `TP=axs.texts[0]`. Potom si zas môžeme pozrieť metódy toho objektu, ktorých je viac ako 100, ale ak sa obmedzíme na metódy, ktoré niečo nastavujú (začínajúce sa na `set` – to je užitočné aj pri iných grafických objektoch), dostaneme po doplnení `TAB`-om asi toto (niektoré riadky sme vynechali):

```
TP.set_alpha                TP.set_ma
TP.set_backgroundcolor      TP.set_name
TP.set_bbox                 TP.set_position
TP.set_clip_on              TP.set_size
TP.set_color                TP.set_style
TP.set_family               TP.set_text
TP.set_ha                   TP.set_x
TP.set_horizontalalignment  TP.set_y
```


Je jasné, že pozíciu, na ktorej sa text vypisuje, nastavíme cez `set_position`, teda pozrieme si o nej help, zistíme, že očakáva usporiadanú dvojicu súradníc a skúsime

```
TP.set_position((1.02,1.62)) # trochu dolava a dole.
```

Keď to urobíme, v obrázku sa nič nezmení. Stačí však (myšou) mierne zmeniť rozmery okna s obrázkom alebo (čo je programátorsky čistejšie) zavolať funkciu `draw()` na prekreslenie aktuálneho obrázka a zmeny sa prejavia.

To už vyzerá celkom dobre, ale chceli by sme, aby sa to písmeno *P* vypisovalo kurzívou (teda nastaviť štýl písma). Môže sa to asi robiť pomocou `set_fontstyle` alebo `set_style`. To druhé je kratšie, skúsime nápovedu a je to ono (povie nám aj to, že možné štýly sú `'normal'`, `'italic'`, `'oblique'`). Teda, stačí prikázať

```
TP.set_style('italic'); draw()
```

a zmena v štýle písma sa prejaví. Skúste si podobne zmeniť veľkosť písma, jeho typ, ale aj text, ktorý sa vypisuje (namiesto *P* dajte napr. *Solution*).

Všimnite si tiež, že obrázok, ktorý máte v tejto knižke je asi trochu iný, než máte vy na obrazovke. Je to tým, že sme spracovanie textu, hlavne matematických výrazov, zverili profesionálnemu sádzaciemu systému LaTeX. Pokiaľ ste na Linuxe a máte nainštalovaný TeX, docieli sa to maličkou zmenou v horeuvedených príkazoch (uvádzame len nové alebo odlišné riadky)

```
rc('text', usetex=True)
...
legend((r'$y=4-2x$',r'$y=x+1$'))
t=text(1.01,1.6,r'$P$')
...
```

To, že sme sa tak dlho venovali nejakému bezvýznamnému písmenku *P*, nebolo náhodou. Chceli sme na ňom ukázať podstatné veci z možností grafiky Pylabu. To, čo platilo o objektoch typu `Text`, bude platiť aj o súradnicových osiach a detailoch ich vykreslenia a hlavne o čiarach a bodoch, z ktorých sa naše grafy budú skladať.

3.2 Ukážky dvojdimenzionálnej grafiky

Príklad 3.2 *Nakreslíme graf funkcie $y = \sin x$ a jej Taylorovych polynómov piateho a desiateho stupňa.*

3 Grafika v Pylabe

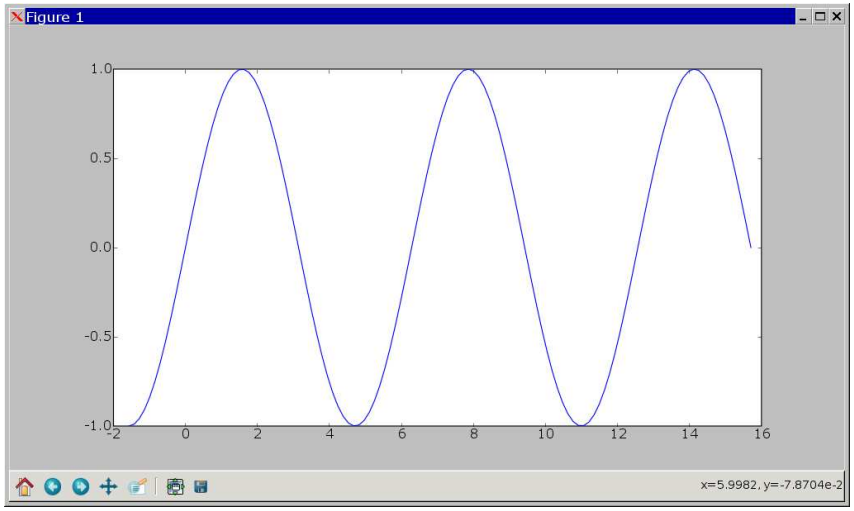
3.1 Základná filozofia, grafické objekty

Grafika v Pylabe je založená na module Matplotlib (HUNTER, 2006). Používame ju dvomi spôsobmi – v dávkovom spracovaní, napr. keď generujeme obrázky pre WEB-server, ktoré my ani neuvidíme, alebo v interaktívnom režime, keď si tie naše obrázky chceme aj prezerať a upravovať.

Základným príkazom pre jednoduché grafy funkcií a kriviek je príkaz `plot`. Nakreslíme si graf funkcie $y = \sin(x)$ pre $x \in \langle -\pi/2, 5\pi \rangle$:

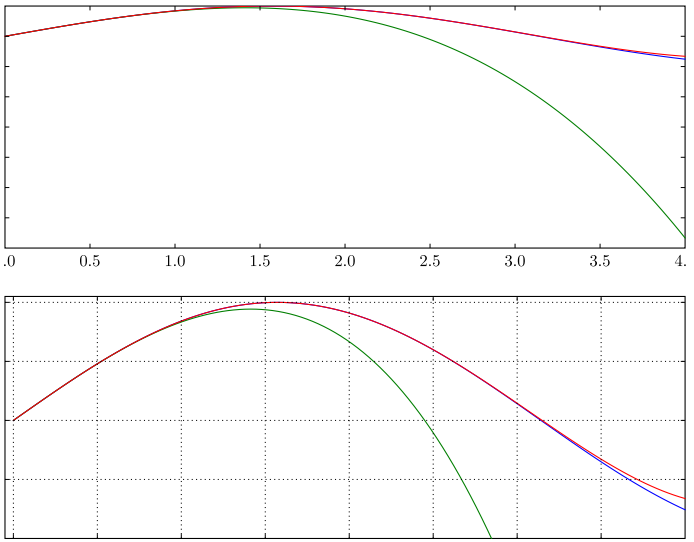
```
x=linspace(-pi/2,5*pi,120)      # hodnoty nezavisle premennej
y=sin(x)                        # funkcne hodnoty v bodoch x
plot(x,y)                       # nakreslenie grafu
```

Na obrazovke sa objaví nové okno, v ktorom bude nakreslený náš graf. Všimnite si v dolnej časti ikony pre interaktívnu prácu s obrázkom. Ikonka domčeka vás vráti k pôvodnému pohľadu, ak ste medzitým urobili nejaký výrez (to sa dá pomocou ikony piatej zľava – tej s lupou). Ikona diskety umožňuje uloženie obrázku v niekoľkých formátoch (napr. `.png`, `.jpg`, `.eps`). Vyskúšajte si tú interaktívnu prácu najlepšie sami. Ak zvolíte zväčšenie výrezu niekoľkokrát, zistíte, že náš graf je vlastne lomená čiara. Je to 120 bodov v rovine (ich súradnice sú určené vektormi `x`, `y`), pospájaných úsečkami. Zatiaľ okno s grafikou neuzavierajte.



čo sme pridali, napr. `matplotlib.text.Text` pre `title`, `text`, `xlabel` alebo `matplotlib.lines.Line2D` pre `plot`. Keď si ich priradíme do nejakých premenných, budeme mať neskôr prístup k ich dátam a metódam, teda budeme ich môcť ľubovoľne modifikovať. Konkrétne, teraz máme v premennej `lines` zoznam troch čiar, ktoré sme do obrázku nakreslili.

Samotný obrázok nám ale neprináša vnútorné uspokojenie, lebo tam vyčíňa puberťácky polynóm piateho stupňa (ponáhľa sa do mínus nekonečna :-) a tým odpútava naše vnímanie od slušného správania sa dospelého polynómu deviateho stupňa.



Pylab robí za nás určenie rozsahu na súradnicových osiach tak, aby sa naše čiary zmestili do grafu. Ale kedykoľvek môžeme tieto rozsahy zmeniť príkazom `axis([xmin,xmax,ymin,ymax])`. V našom prípade dolnú hranicu na osi `x` dáme troška pod nulu (aby sa nezlievali čísla pri popisoch osí) a rozsah na osi `y` nastavíme od -1 do 1.05 (aby krivky neboli celkom na hornom okraji obrázka).

Na aktuálny obrázok pridáme aj sieťku pre ľahšie odčítanie súradníc. Funkcia `grid()` bez parametrov je prepínač – ak nebola sieťka, bude a naopak, ale môžeme aj explicitne špecifikovať napr. `grid(True)`.

```
axis([-0.05, 4, -1, 1.05])
grid()
```

who, že už ju nenájdete medzi živými, t. j. existujúcimi premennými. Načítame ju ale znova z nášho súboru príkazom

```
Zh_nove=load('gr12_53.txt')
```

Výsledkom bude číselné pole s 53 riadkami a dvomi stĺpcami; každý riadok je jedna hrana. Ale keď si dáte niektorý riadok vypísať, zistíte, že jeho dva prvky sú reálne a nie celé čísla. Našťastie, existuje metóda na pretypovanie celého poľa (volá sa astype) a pomocou nej dostaneme už celočíselné pole hrán Zh (to isté, ako sme mali na začiatku):

```
Zh=Zh_nove.astype(int)
```

Ak je dát veľa, oplatí sa uschovávať ich v binárnom formáte, ktorý je pre ľudí nečitateľný, ale výsledný súbor je menší a načíta sa do počítača nepomerne rýchlejšie, ako textový súbor s ekvivalentným obsahom.

Ukážeme si to na náhodnej matici 1000×1000 s celočíselnými prvkami, ktorú nagenerujeme príkazmi:

```
import scipy
# jednorozm. pole s milion prvkami v rozsahu 1-157
R=scipy.stats.randint.rvs(1,157,1e6)
R.shape=(1000,1000)  # premenime ho na maticu 1000x1000
```

Na vysvetlenie – v module scipy.stats je veľa spojitých i diskrétnych rozdelení náhodnej premennej. Každé rozdelenie má ešte rôzne metódy, napr. rvs (random variable samples), t. j. generovanie náhodných vzoriek z tohto rozdelenia, pdf – hustota pravdepodobnosti, cdf – distribučná funkcia alebo pmf (probability mass function), teda pravdepodobnostná funkcia diskrétneho rozdelenia, atď. My sme využili rovnomerné diskrétne rozdelenie randint pre celé čísla v intervale $\langle 1, 157 \rangle$, z ktorého sme urobili náhodný výberový súbor s milión prvkami.

Každé číselné pole má metódu tofile, pomocou ktorej ho môžeme binárne zapísať do súboru. V našom prípade urobíme:

```
# otvorime subor s nazvom Rbig.bin na zapisovanie
outfile=file("Rbig.bin","w")
R.tofile(outfile)             # zapiseme donho maticu R
outfile.close()               # zatvorime subor, schluss
```

Textový súbor, ktorý by sme dostali príkazom save("Rbig.txt",R,"%i") je síce trocha menší ako binárny súbor, ale zápis do binárneho súboru je viac ako osemdesiatkrát rýchlejší. Pri načítavaní dát ukážeme, ako sme to merali. Nezabudnite, že súbor, do ktorého zapisujete, treba po skončení zápisu a pred ďalšími manipuláciami s ním (obyčajne čítaním) uzavrieť

Príklad 3.3 *Majme 40 dátových bodov (x_i, y_i), ktoré sú výsledkom meraní. Vieme, že teoreticky by mala byť medzi nimi lineárna závislosť $y = ax + b$. Urobme pre tieto dáta priamkovú aproximáciu pomocou metódy najmenších štvorcov (MNŠ) a nakreslime pôvodné body aj aproximačnú priamku.*

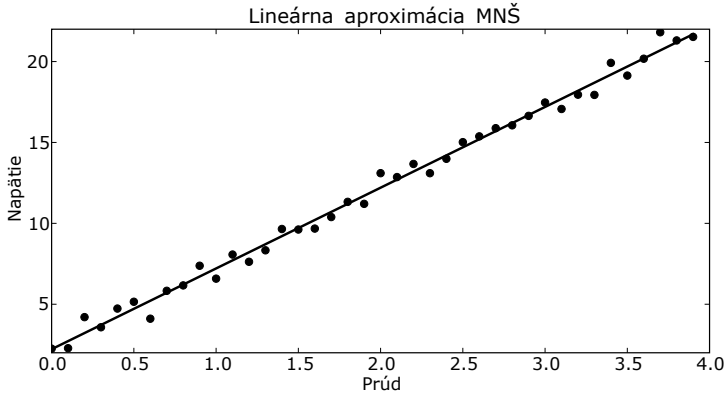
Tie dáta simulujeme pomocou malých náhodných odchýliek s normálnym rozdelením. Na aproximáciu použijeme funkciu polyfit, ktorá robí polynomickú aproximáciu pomocou MNŠ. Dátové body nakreslíme ako malé čierne kruhy a priamku dáme trocha hrubšiu:

```
from pylab import *

x=arange(0,4,0.1)      # 40 bodov na osi x
odch=0.5*randn(len(x)) # nahodne odchylky
y=2+5*x+odch           # linearne data s chybami
a,b=polyfit(x,y,1)     # posledny argument je stupen polyn.
plot(x,y,'ko',x, a*x+b,'-k',lw=3)
```

Ten titulok grafu a popis osí s diakritikou sme vyrobili pomocou príkazov

```
title(unicode("Lineárna aproximácia MNŠ","latin2"))
xlabel(unicode("Prúd","latin2"))
ylabel(unicode("Napätie","latin2"))
show()        # aby sa zmeny v obrazku prejavili
```



Nové v príkaze plot je, že sme použili štýl bodov na kreslenie dátových bodov (inak by to Pylab pospájal plnou lomenou čiarou), a že sme pri znázorňovaní priamky použili pomenovaný argument lw (skratka za linewidth) na nastavenie hrúbky čiary.

Príklad 3.4 *Nakreslíme ilustráciu k tomuto príkladu: „Zistite plochu rovinnej oblasti, ohraničenej parabolami $y_1 = x^2$, $y_2 = x + 3 - x^2$.“*

```
    else:
        a_akt,a_predch=1,0
        for k in range(2,n+1):
            a_akt,a_predch=a_akt+a_predch,a_akt
        return a_akt
```

Nové je v týchto funkciách ošetrenie vstupu tak, aby to neprešlo, ak vstup nie je celé číslo a ešte k tomu aj nezáporné. Funkcia `assert` zaistí, aby v nej uvedené predpoklady boli splnené. Ak nie sú, generuje sa chyba a môže sa prípadne vypísať objasňujúci text.

Podobne ako operátor `and` aj ostatné logické operátory `or`, `not` sú pomenované príslušnými anglickými slovami (ale ako sme povedali už vyššie, pre číselné polia používajte `&`, `|`, `~`).

V tom príklade vidíte tiež jednoduché vetvenie programu podľa podmienky – pre n patriace do zoznamu `[1,2]` je faktoriál rovný jednej, inak sa počíta v cykle opakovaným násobením. Rozmyslite si, že v našom príklade tú podmienku `else` ani netreba. Všeobecný príkaz na vetvenie môže obsahovať ešte niekoľko vetiev, začínajúcich kľúčovým slovom `elif`.

2.2 Práca s dátovými súbormi

Na uschovanie dát v textových súboroch a ich opätovné načítanie máme dvojicu funkcií `save`, `load`.

Príklad 2.4 *Vytvorme náhodný neorientovaný graf, ktorý má dvanásť vrcholov a 53 hrán. Zoznam hrán zapíšme do súboru a potom tie hrany z neho načítajme.*

Graf sa tu chápe v zmysle dopravnej siete (cestnej, železničnej), t. j. ako množina uzlov (vrcholov), pospájaných cestami (hranami). Vytvorenie grafu bude väčší problém než tie súbory. Poďme cestou najmenšieho odporu a hrubou silou. V module `numpy.random` (keďže je to submodul pre pravdepodobnosť a štatistiku, je logické, že hľadáme tam) nájdeme našim obľúbeným doplňovaním klávesom `TAB`, že existuje funkcia `numpy.random.random_integers` a tá sa nám hodí. Keďže nám nerobí problém napísať všeobecnú funkciu pre náhodný graf s `n` vrcholmi (očíslujeme ich poradovými číslami od 1 do `n`) a `m` hranami, urobíme to. Uložte si ju do súboru `randgraph.py`. V tej funkcii budeme generovať hrany (dvojice čísel 1 až `n`, pričom na poradí nezáleží, teda vždy môžeme zobrať prvý prvok menší, ako druhý), kým ich nebude presne `m`:

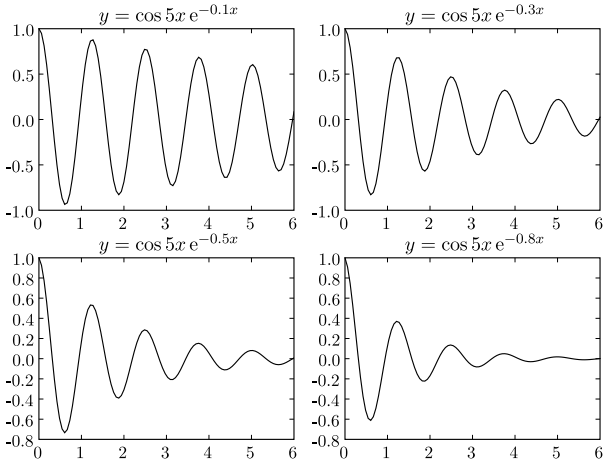
```
from numpy.random import random_integers as rndint
def randgraph(n,m):
```



```
xvel=linspace(-1.5,2.0,140)
yv1,yv2=xvel*xvel,xvel+3-xvel*xvel
plot(xvel,yv1,'-.k',xvel,yv2,'--k')
legend((r"$y_1=x^2$", r"$y_2=x+3-x^2$"),loc=(0.68,0.86))
title("Plocha rovinnej oblasti.")
```

Zaujímavé je tu manuálne umiestnenie legendy. Keby ste nedali parameter `loc`, legenda sa umiestni v pravom hornom rohu a prekryje jednu z parabol. Experimentovaním sme prišli k vhodným súradniciam tak, aby legenda ničomu nezavadzala. Pozor, sú to vždy „normalizované súradnice“ od 0 do 1 na oboch osiach.

Príklad 3.5 *Nakreslíme do jedného grafu štyri krivky pre tlmené kmitanie, dané rovnicou $y = \cos(5x) \cdot \mathrm{e}^{-\alpha x}$ s tlmiacimi činiteľmi $\alpha = 0.1, 0.3, 0.5, 0.8$.*



To sa robí pomocou príkazu `subplot`. Keď chceme vedľa seba `m` obrázkov, ktoré sú usporiadané v `n` riadkoch, dáme príkaz

```
subplot(m,n,k)
```

. Ten nás zároveň nastaví na `k`-ty obrázok (počíta sa po riadkoch smerom odhora a v riadku zľava doprava).

```
x=linspace(0,6,100)
y1=cos(5*x)*exp(-0.1*x); y2=cos(5*x)*exp(-0.3*x)
subplot(2,2,1);  plot(x,y1,'k')
title(r"$y=\cos 5x\,{\rm e}^{-0.1 x}$")
subplot(2,2,2);  plot(x,y2,'k')
title(r"$y=\cos 5x\,{\rm e}^{-0.3 x}$")
```

Keď chceme, aby naša funkcia vrátila nejaký objekt, musíme to explicitne povedať príkazom `return`. Keď to neurobíme, funkcia vráti „Pythoňácke nič“, čo je špeciálny objekt `None`. Samozrejme, z funkcie môžeme vrátiť aj viac objektov naraz, napr. cez usporiadanú n-ticu ($n \geqq 2$).

V Pythone funguje viacnásobné priradenie, napr. `a=b=1`, alebo podobne `a,b=1,2`, t. j. usporiadanej dvojici `(a,b)` sa priradia príslušné hodnoty z číselnej dvojice `(1,2)`. Dva objekty zameníme jednoducho príkazom `a,b=b,a`. Všimnite si, že usporiadaná n-tica to sú objekty, oddelené čiarkou. Písať okrúhle zátvorky niekedy nie je nutné.

Často používaným trikom je „rozdistribuovanie“ výsledku funkcie do viacerých premenných. Napr. ak `MP="Alžbeta Kalininova"` je textový reťazec, tak príkazom

$$\texttt{meno,priezvisko} = \texttt{MP.split}()$$

urobíme priradenia `meno="Alžbeta"`, `priezvisko="Kalininova"`. Pozrite si, čo robí metóda `split` pre objekty typu `str`.

V Pylabe môžeme pohodlne zapisovať aj „intervalové“ nerovnosti, napr.

$$-1 \leqq x < 3 \text{ zapíšeme ako } -1 <= \texttt{x} < 3.$$

Zaujímavý je prvý riadok vyššieuvedeného súboru `hilbert.py`. V Pythone (podobne, ako je to v C-čku) je väčšina funkcionality prístupná v **moduloch**, čo je obdoba PASCAL-ovských „units“ alebo C-čkovských knižníc. Napr. funkcia `empty` je v module `numpy` a sprístupníme ju (importujeme do nášho programu) práve príkazom: `from numpy import empty`. Naraz môžeme z toho istého modulu importovať aj viac objektov, napr.:

```
from numpy import zeros, ones, eye, empty, dot.
```

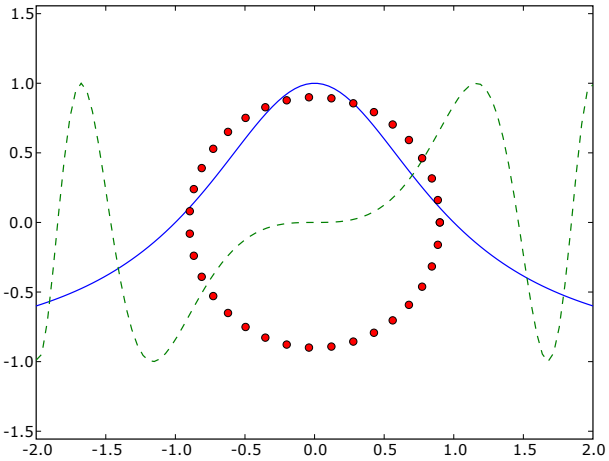
Ako vieme, v ktorom module (resp. submodule) je naša funkcia? Nuž, vyskúšame moduly, tvoriace Pylab v tomto poradí `numpy`, `scipy`, `pylab`. Základné funkcie na prácu s maticami a vektormi budú v module `numpy`, tie druhé dva sú nadstavbou nad týmto fundamentálnym modulom. V interaktívnom prostredí importujeme oba výpočtové moduly, teda `import numpy, scipy`. Predpokladajme, že chceme vedieť, odkiaľ treba importovať funkciu `solve`. Skúsime `numpy.sol` a tiež `scipy.sol`, či nám to doplní `TAB`-om. Ak nie, tak tam tá funkcia nie je.

Ak niečo nie je v moduloch `numpy` ani `scipy`, treba vyskúšať submoduly v `scipy`. Najužitočnejšie pre nás budú submoduly `scipy.linalg`, `scipy.interpolate`, `scipy.integrate`, `scipy.optimize`, `scipy.stats`. V našom prípade zaberie `scipy.linalg.sol`, lebo to sa nám už `TAB`-om


```
fi=linspace(0,2*pi,37)          # prečo nie 36?
xt=0.9*cos(fi); yt=0.9*sin(fi)
plot(xt,yt,'or')
```

Všimnime si, že kružnica je na obrazovke akási pretiahnutá v smere osi y, t. j. je to vlastne elipsa. Keď chceme, aby sa jednotkové dĺžky na osiach zobrazovali rovnako (na obrazovke, ale aj po vytlačení obrázku, o čom sa môžete presvedčiť na vlastné oči alebo pomeraním), použijeme príkaz

```
axis("equal")
```



Ten istý príkaz v tvare `axis([xmin, xmax, ymin, ymax])` môžeme použiť na manuálne nastavenie rozsahov na súradnicových osiach, ak nám nevyhovuje to, čo Pylab vyberie automaticky.

Krivky v polárnych súradniciach (teda v tvare $\rho = \rho(\varphi)$, $\varphi \in \langle \alpha, \beta \rangle$) môžeme kresliť pomocou príkazu `polar`. Po vyčistení obrázka pomocou `clf()` môžeme tú predchádzajúcu kružnicu nakresliť príkazom

```
polar(fi,[0.9]*len(fi),'or')
```

Druhý argument `[0.9]*len(fi)` je zoznam, ktorého každý prvok je 0.9 a má dĺžku rovnakú ako vektor `fi`.

Cvičenie 3.7 *Elipsu s poloosami $a = 2$, $b = 1$ môžeme nakresliť v polárnych súradniciach, ak si uvedomíme, že jej parametrická rovnica je*

$$x = a\cos\varphi,\ y = b\sin\varphi, \quad \varphi \in \langle 0, 2\pi \rangle$$

a že polárny sprievodič $\rho(\varphi)$ sa dá určiť ako $\rho = \sqrt{x^2 + y^2}$.

2 Programovanie v Pylabe

2.1 Základy na prežitie

Pylab je interaktívny interpretačný jazyk. Vy mu zadávate rôzne príkazy (v príkazovom riadku na obrazovke), on ich vykonáva. Keď však chcete robiť komplikovanejšie veci, napíšete si váš program s pomocou nejakého textového editora (my budeme používať `nedit`, `http://www.nedit.org/`, čo je skratka z Nirvana Editor :-), uložíte na disk a potom ho spustíte v našom interaktívnom prostredí cez príkaz `run`.

Programovanie v Pylabe je vlastne programovaním v Pythone. Takže máme k dispozícii všetky dátové typy (čísla celé, reálne a komplexné, textové reťazce, zoznamy, usporiadané n-tice, asociatívne polia) a príkazy (vrátane cyklov, podmienok, ošetrovania výnimočných situácií) z Pythonu. Pre produktívnu prácu v Pylabe však stačí veľmi málo a to si ukážeme na jednoduchých príkladoch.

Príklad 2.1 *Napíšme funkciu, ktorá vytvorí Hilbertovu maticu $n \times n$, t. j. maticu s prvkami $h_{ij} = 1/(i+j-1)$, $i, j = 1, 2, \ldots, n$.*

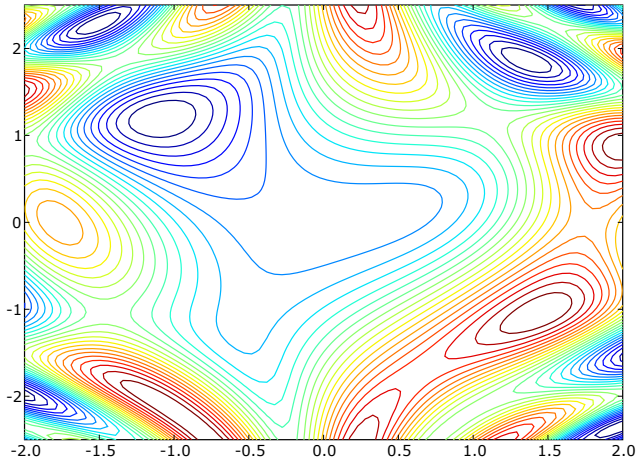
Najskôr musíme správne nakonfigurovať editor, v ktorom budeme naše zdrojové programy písať. Je to veľmi dôležité, pretože Python používa indentáciu (odsadenie textu) na označenie blokov v programe (teda to, čo sa v PASCALe značí pomocou `begin`, `end` a v C-čku pomocou zložených zátvoriek). **Editor treba nastaviť tak, aby namiesto tabelátorov vkladal štyri medzery** (ak to vo vašom editore nejde, prejdite na iný :-). A potom **na indentáciu zásadne využívajte kláves TAB**, nie medzery. Ak ešte nemáte vybraný vhodný editor, skúste si pozrieť `http://stani.be/python/spe/blog/`, editor, či skôr IDE SPE je napísaný v Pythone a je veľmi dobrý.

Ak sa budete týchto rád držať, nebudete mať žiadne problémy. Ak nie, dočkáte sa mnohých hlášok typu `Indentation error` a budete nadávať na Python. Najhoršie, čo môže byť, je zdroják, v ktorom sú pomiešané tabulátory a medzery v indentácii. V každom editore to môže potom vyzerať inak. Čo je horšie, úplne sa môže zmeniť zmysel zdrojáku. Autor to môže potvrdiť z vlastnej skúsenosti, keď začínal s Pythonom veľmi krvopotne práve z uvedených malicherných dôvodov.

Takže predpokladáme, že máte editor správne nastavený. Dajte si vytvoriť nový súbor, ktorý hneď pomenujte napr. `hilbert.py`. To ukončenie súboru na `.py` pomôže editoru, aby nastavil zvýrazňovanie syntaxe, príp. doplňanie kľúčových slov, a pod.

V našom malom súbore vidíte definíciu funkcie, potom inicializáciu matice (alokáciu pamäte). Ďalej sú dva vnorené príkazy cyklov a nako-

sme dosiahli ďalšími argumentami (týka sa to popisu osí a počtu vrstevníc).



Existuje príkaz `contourf`, ktorý priestor medzi vrstevnicami vyplní farebne, podobne ako to býva na mapách. Pomocou príkazu `colorbar` tiež môžeme pridať farebnú stupnicu ku grafu, takže vieme, aká farba prislúcha danej veľkosti znázorňovanej veličiny.

Príklad 3.9 *Pre plochu $f(x,y) = |p_4(x+iy)|$, kde $z = x+iy$,*

$$p_4(z) = z^4 - 2z^3 + 2z^2 + 5z - 3, \quad -1.3 \leqq x \leqq 1.4, \quad -1.7 \leqq y \leqq 1.7$$

nakreslite farebne vyplnený vrstevnicový graf. Voľte vhodné úrovne pre vrstevnice tak, aby ste získali dobrú predstavu o tej ploche a tiež aj o polohe komplexných koreňov polynómu $p_4(z)$.

Polynóm $p_4(z)$ má koreň $z_0 = x_0 + iy_0$ práve vtedy, keď $f(x_0+iy_0) = 0$. Postupujeme podobne, ako v predchádzajúcom príklade:

```
from numpy import polyval
xx,yy=linspace(-1.3, 1.4,80),linspace(-1.7, 1.7,80)
X,Y=meshgrid(xx,yy)
j=complex(0,1)                  # imaginarna jednicka, 0+1j
Z=X+j*Y                         # sietka v komplex. rovine
c4=[1,-2,2,5,-3]                # koeficienty polynomu
Fxy=abs(polyval(c4,Z))          # vypocet f(x,y) na sietke
```

```
A=rand(100,100)  # nahodna matica 100 x 100
A7=(A>0.7)       # bool. matica, True kde a[i,j] > 0.7
P7=A7.sum()      # sucet prvkov A7 je ten pocet
                 # (True -> 1, False -> 0 v sucte)
```

Všimnite si, ze metóda `sum` sčíta všetky prvky daného číselného poľa (aj viacrozmerného). Ak potrebujeme vektory stĺpcových alebo riadkových súčtov, treba zadať ešte „súradnicovú os“ – 0 pre stĺpcové a 1 pre riadkové súčty, teda `S_riad=A.sum(0); S_stlp=A.sum(1)`.

1.4 Základne funkcie pre prácu s polynómami a maticami

Spomenieme niekoľko funkcií pre polynómy a matice, ktoré asi budete často používať.

Polynóm $P_n(x) = c_n x^n + c_{n-1}x^{n-1} + \cdots + c_1 x + c_0$ je v Pylabe reprezentovaný vektorom jeho koeficientov, pričom začíname od najvyššej mocniny premennej x. Napr. $P_5(x) = x^5 - 2x^3 + 7x^2 - x - 3$ je daný vektorom `c=[1, 0, -2, 7, -1, -3]`.

Teda žiadne „ručné“ programovanie mocnín premennej a pod. Na počítanie hodnôt polynómu používame funkciu `polyval`, takže

$$\texttt{y5} = \texttt{polyval}(\texttt{c}, \texttt{rand}(100))$$

uloží do premennej `y5` hodnoty horeuvedeného polynómu, vypočítané v 100 náhodných bodoch.

Dvojica funkcií `roots`, `poly` slúži na výpočet koreňov polynómu a opačne, na rekonštrukciu koeficientov polynómu, ak sú dané jeho korene. Vyskúšajme si (príkaz `from scipy import poly` na prvom riadku slúži na načítanie funkcie `poly` z modulu `scipy` a vysvetlíme ho podrobnejšie v ďalšom odseku):

```
from scipy import poly
W=poly(arange(1.0,21.0)) # Koeficienty Wilkinsonovho
                         # polynomu s korenmi 1,2,...20
R=roots(W20)             # korene toho polynomu
                         # to, co uvidite, je zivot :-)
```

Na násobenie a delenie polynómov máme funkcie `polymul`, `polydiv`. Vynásobme polynómy $p = (x-1)^5$, $q = x^2 + 5x - 6$ a súčin vydelíme opäť polynómom q, čím by sme mali dostať polynóm p a zvyšok 0. Je to tak?

```
from scipy import polymul, polydiv
cp=poly([1.0]*5)      # polynom p s 5-nasob. korenom 1
```

4 Ukážky použitia Pylabu v numerike

4.1 Riešenie sústav nelineárnych rovníc

Keď sme prestali u vrstevnicových grafov, tak teraz tam začneme – úlohou o riešení sústavy dvoch nelineárnych rovníc o dvoch neznámych, pretože sa dá veľmi pekne vizualizovať. Majme teda sústavu rovníc

$$\begin{aligned} z_1 &= f_1(x,y) = 0, \\ z_2 &= f_2(x,y) = 0. \end{aligned}$$

Týmito rovnicami sú určené dve plochy z_1 a z_2. Riešiť zadanú sústavu znamená nájsť spoločné body (priesečníky) nulových vrstevníc týchto plôch a tie vieme znázorniť pomocou vrstevnicového grafu.

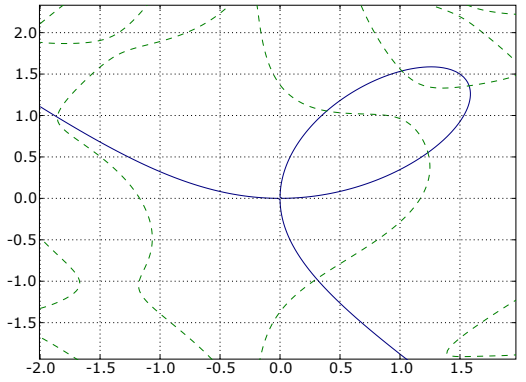
Príklad 4.1 *Nájdime tie riešenia sústavy rovníc*

$$\begin{aligned} \sin(xy^2) - \cos(x^2 - y) + 0.2 &= 0, \\ x^3 + y^3 - 3xy &= 0, \end{aligned}$$

ktoré ležia v prvom kvadrante (t. j. majú obidve súradnice kladné).

Všimnite si, že vrstevnicový graf podobnej funkcie sme robili v príklade 3.8. Budeme postupovať ako tam, len do toho istého grafu nakreslíme nulovú vrstevnicu aj pre druhú funkciu. Hladiny vrstevníc sa vždy musia zadávať ako zoznam (aj keď len jednoprvkový), inak by to Pylab chápal ako počet vrstevníc.

```
xx=linspace(-2,2,120);      yy=linspace(-2,2,120)
X,Y=meshgrid(xx,yy)
Z1=sin(X*Y*Y)-cos(X*X-Y)+0.2; Z2=X**3+Y**3-3*X*Y
contour(X,Y,Z1,[0.0]);      contour(X,Y,Z2,[0.0],colors='g')
```



1.3 Vyberanie a priraďovanie prvkov a submatíc, operátory porovnávania a ich využitie

Jednotlivé prvky vyberáme z matice pomocou indexovania hranatými zátvorkami, ako je to zvykom aj v iných programovacích jazykoch. Teda to, čo matematicky zapíšeme ako $\mathbf{A}_{ij}$, budeme v Pylabe písať ako `A[i,j]`. Je dôležité vedieť, že číslovanie riadkov a stĺpcov začína od nuly (tak, ako je to v jazyku C), takže prvok v „ľavom hornom rohu“ matice `A` je `A[0,0]`. Celé riadky vyberáme zadaním jedného indexu, napr. `A[0]` je prvý riadok matice `A`.

Namiesto číselných indexov môžu byť tzv. *slices* (po slovensky hádam „krajce“), ktoré vyzerajú ako

$$\texttt{i\_beg} : \texttt{i\_end} : \texttt{k},$$

kde `i_beg` je začiatočný index, `i_end` koncový index a `k` je krok. Táto dvojbodková syntax je ekvivalentná príkazu `arange(i_beg, i_end+1, k)`, takže platí všetko, čo sme povedali vyššie o tomto príkaze. Ibaže krajce sa dajú použiť len pri indexovaní polí. Ukážeme to na príkladoch, kde predpokladáme, že `A` je matica 5×5, generovaná napr. príkazom `A=rand(5,5)`

```
A[:2,:2]      # submatica - prve dva riadky a stlpce
A[1::2,::2]   # riadky 1,3 a stlpce 0,2,4
A[1:3,3]      # riadky 1,2 a stlpec 3
A[:,2]        # vsetky riadky, stlpec 2
A[::-1,:]     # riadky matice A v opacnom poradi
A[[3,0,2],:]  # riadky 3,0,2 a vsetky stlpce
```

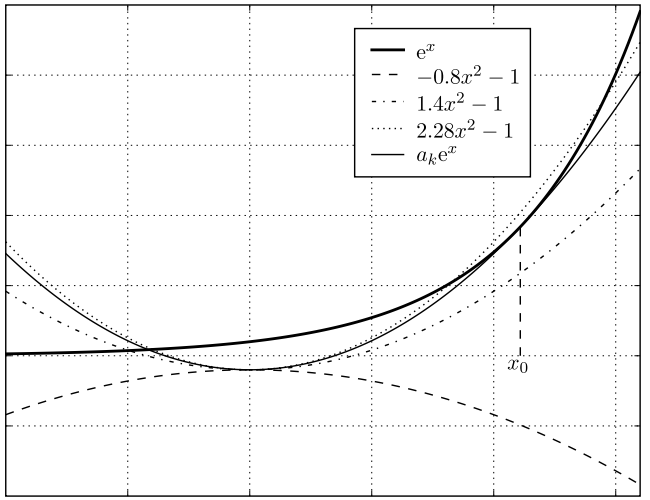
Myslíme, že z uvedených príkladov je jasné, čo sa stane, keď sa vynechá začiatočný index, koncový index či krok. Osobitne zaujímavé je vytvorenie vektora s opačným poradím prvkov príkazom `v[::-1]`.

Výber prvkov pomocou „dvojbodkového označenia“ môžeme robiť na oboch stranách priraďovacieho príkazu, napr.

$$\texttt{A}[:, [1, 2, 4]] = \texttt{B}[:, 1 : 4]$$

nahradí stĺpce 1, 2, 4 matice `A` prvými tromi stĺpcami matice `B`. Pri podobnom priraďovaní treba dbať na rozmerovú kompatibilitu (pravá a ľavá strana musia mať ten istý rozmer). Na pravej strane môže byť tiež len jedno číslo, takže príkazom

$$\texttt{A}[: 2, : 3] = 0$$



Z obrázka je vidieť, že pre $a \leqq 0$ neexistuje žiadny reálny koreň, pretože vtedy $ax^2 - 1 \leqq -1$, kým $e^x > 0$ pre všetky reálne x.

Pre $a > 0$, ak a je dostatočne malé, rovnica má jeden záporný koreň. Ak však a bude dostatočne veľké, budeme mať tri korene (jeden záporný a dva kladné). Najdôležitejší je pre nás „medzný prípad“, kedy pre nejaké $a = a_k$ má parabola a exponenciála spoločnú dotyčnicu. Označme x-ovú súradnicu dotykového bodu ako x_0. Potom na určenie a_k, x_0 máme sústavu nelineárnych rovníc, ktorá vyplýva z rovnosti funkčných hodnôt a tiež derivácií v bode x_0

$$\begin{aligned} a_k x_0^2 - 1 &= e^{x_0} \\ 2a_k x_0 &= e^{x_0}. \end{aligned}$$

Z druhej rovnice ľahko vyjadríme $a_k = e^{x_0}/(2x_0)$ a dosadením do prvej dostaneme pre x_0 rovnicu (index nepíšeme, je zbytočný)

$$e^x(x - 2) - 2 = 0.$$

Tú ľahko vyriešime v Pylabe a dopočítame príslušný parameter a_k:

```
f=lambda x: exp(x)*(x-2)-2   # nelin. rovnica pre x0
x0=fsolve(f,1.5)             # x0=2.217715105757
ak=exp(x0)/(2*x0)            # ak=2.071122003228
```

Skrátenú definíciu funcie `f` pomocou tzv. lambda notácie vysvetlíme podrobnejšie v nasledujúcom príklade. Takže, ako zhrnutie výsledkov, môžeme povedať, že pre $0 < a < a_k$ má pôvodná rovnica jeden reálny koreň, pre $a = a_k \doteq 2.071122$ má dva korene a pre $a > a_k$ tri korene.

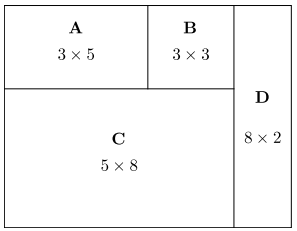
pretože **Pylab inteligentne doplňuje názvy objektov a ich metód**. Používame na to kláves `TAB` – to je ten s dvomi diódami :-). Konkrétne, ak napíšete `A.` (tá bodka je tam dôležitá) a stlačíte `TAB`, vypíšu sa vám na obrazovku všetky dáta a metódy objektu `A`. Keď je toho veľa (akože aj je), napíšeme napr. `A.tra` a potom stlačíme `TAB`. A dostaneme len dve metódy `A.trace`, `A.transpose`. Ak nám nie je intuitívne jasné z názvu (alebo experimentovaním), o čom daná vec je, dáme si nápovedu, napr. príkaz `A.trace?`.

Teraz prezradíme, že `A.shape` je usporiadaná n-tica, určujúca rozmery poľa `A`. Konkrétne, pre maticu `A` z nášho príkladu nám Pylab vráti dvojicu `(2,2)`. Vyskúšajte, čo dostanete pre vektory `B`, `Br`, `Bs` a maticu `C`. Potom vám bude jasné, prečo nejde urobiť maticový súčin `dot(C,Br)`. Dá sa urobiť `dot(C,B)` a prečo?

Je zaujímavé, že rozmery matice môžeme meniť kedykoľvek, priradením do jej dátového atribútu `shape`, teda ak dáme `C.shape=(6,2)`, naraz sa z matice 4×3 stane matica 6×2. Rozmyslite si, ako by ste týmto štýlom z ľubovoľnej matice urobili jednorozmerné pole (malá nápoveda: `(m,n)=C.shape` priradí do premennej `m` počet riadkov a do premennej `n` počet stĺpcov matice `C`).

Je jasné, že zadávanie matíc prvkami sa hodí len pre nie príliš rozmerné matice. Numerická analýza sa však používa na riešenie reálnych problémov, kde matice rádove 100×100 sú „malé“. Našťastie, „**prvkami“** pri zadávaní matice možu byť nielen čísla, ale tiež celé **maticové bloky** či **submatice**. Matice v aplikáciach majú často blokovú štruktúru a v Pylabe máme na ich konštrukciu funkciu `bmat`.

Napríklad matica `M` typu 8×10 môže byť „pozliepaná“ z menších blokov `A`, `B`, `C`, `D`:




```
A=ones((3,5))
B=zeros((3,3))
C=3*ones((5,8))
D=-ones((8,2))
P=bmat('A,B;C'); M=bmat('P,D')
```

Najskôr vytvoríme pomocnú maticu `P` typu 8×8, ktorej prvý „riadok“ je tvorený maticami `A`, `B` a druhý „riadok“ je `C`. Potom k `P` pridáme sprava maticu `D`. Všimnite si, že argument pre funkciu `bmat` zadávame ako textový reťazec; v ňom sú prvky (bloky) v riadku oddelené čiarkami a riadky sa oddeľujú bodkočiarkou (tak je to zvykom v MATLAB-e). Vidíte tiež,

môžeme vypočítať aj príslušný určitý integrál vo vyjadrení funkcie $P(r)$. Vieme ho síce spočítať aj vzorcom, ale urobíme numerický výpočet pomocou funkcie `quad` zo submodulu `scipy.integrate`. Na riešenie rovnice $P(r) - 10 = 0$ použijeme funkciu `fsolve`, ktorú poznáme z predchádzajúceho príkladu. Výpočet v Pylabe realizujeme pomocou súboru `koza.py`:

```
from numpy import roots, sqrt
from scipy.optimize import fsolve
from scipy.integrate import quad

Pr_forint=lambda x,r: sqrt(r**2-(x-1)**2)-3-x**2
TOL=1.0e-8

def Pr10(r):
    c=[1, 0, 7, -2, 10-r*r]
    rt=roots(c)
    rr=rt[abs(rt.imag)<TOL].real
    if not len(rr):
        raise ValueError, "Rope too short, goat passed!"
    rr.sort()
    Ip,err=quad(Pr_forint,rr[0],rr[1],args=r)
    return Ip-10

print "Optimal Rope length: ", fsolve(Pr10,8)
```

Pomerne jednoduché, však? V MATLAB-e by to bolo trochu komplikovanejšie. Vysvetlíme si, čo sa v tom súbore robí. Riadok

```
Pr_forint=lambda x,r: sqrt(r**2-(x-1)**2)-3-x**2
```

definuje funkciu, ktorá sa vyskytuje v integrále. Áno, `Pr_forint` je funkcia, definovaná pomocou tzv. lambda notácie. Hodí sa to, keď máte jednoduchý výraz, ktorý funkcia vracia, napr. môžete si nadefinovať

```
pythag=lambda x,y: sqrt(x**2+y**2);   sqr=lambda(x): x*x
```

a potom tie funkcie normálne voláte – skúste si `pythag(3,4); sqr(-2); sqr(10)` a podobne. Samozrejme, zložitejšie funkcie je lepšie definovať pomocou kľúčového slova **def** ako sme to robili doteraz. Takto je definovaná aj funkcia `Pr10`, ktorá pre dané r určí hodnotu $P(r) - 10$.

V premennej `rt` sú korene polynómu (4.1). Tie sú buď všetky komplexné, ak je špagát príliš krátky (napr. $r = 2$; vtedy generujeme chybu s hláškou o skapíňajúcej koze) alebo dva z nich sú reálne a to sú hranice $a(r)$, $b(r)$ pre integrál. Zaujímavý je príkaz

```
Bs= array([[1],[-2],[3]])    # stlpcovy vektor
C = rand(4,3)        # nahodna mat., rovnomer. rozd. [0,1)
D = zeros((3,3),'d') # matica nul (realnych, nie int.)
F = 4*ones((4,2))    # matica zo samych stvoriek (int.)
E = eye(4)           # jednotkova matica 4x4 (matematicka)
G = empty((5,5))     # neinicializovana mat. 5x5, rychle
Am= mat('1,2,4;4,5,6;7,8,9')
```

Možno ste pri zadávaní príkazov zmätení, že nevidíte nič na obrazovke. Príkaz vykonáte a akoby sa nič nestalo? No, stalo sa. Do premenných `A`, `B`, ..., `G`, `Am` máte priradené, to, čo ste zadali. Výpis premennej na obrazovku zariadite jednoducho napísaním jej mena alebo cez príkaz `print`, napr. `print(A)`. Keby ste niečo vytvorili, ale nikde nepriradili, automaticky sa to vypíše na obrazovku – to je najrozumnejšie, čo môže systém pre vás urobiť (samozrejme, nepriradené objekty nemôžete použiť v ďalších výpočtoch).

Väčšina z vás už asi uhádla, že texty za znakom `#` (až do konca riadku) v horeuvedených príkladoch sú komentáre. Teda sú to vysvetľujúce poznámky pre nás, ľudí a Pylab ich ignoruje. Je dobre si robiť komentáre, sú trvácnejšie, ako poznámky v zošite :-). Vaše zdrojáky a tiež históriu interaktívnych príkazov si môžete kedykoľvek pozrieť na počítači, na ktorom robíte, alebo si ich stiahnuť cez sieť.

Často, hlavne keď už robíte v Pylabe dlhší čas, stratíte prehľad o tom, čo za premenné a funkcie ste si vyrobili. Takže namiesto toho, aby ste si lámali hlavu, či sa vaša matica volá `A`, `a` alebo `Aa`, **spýtajte sa systému – Hej, kto je tu?** – príkazom `who`. Ufrflanejší príkaz `whos` vám povie aj, aké typy majú vaše objekty a vypíše aj rozumné množstvo ich prvkov (ak sú to objekty sekvenčného typu). Ľubovoľný objekt môžete zlikvidovať príkazom `del`, napr. `del(G)` alebo pre viaceré objekty `del(Ob1,Ob2,Ob3)`. Ale tie matice, čo sme vytvorili vyššie, si nechajte, budeme sa na ne ešte odvolávať. Treba si zvyknúť tiež na to, že Pylab rozlišuje malé a veľké písmená, teda `AA`, `Aa`, `aA`, `aa` sú štyri rôzne mená objektov v Pylabe.

Asi ste pochopili, že príkaz `array` vytvorí jedno alebo viacrozmerné pole, ak mu zadáte nejaký *zoznam* (to sú tie veci v hranatých zátvorkách). No a `rand`, `zeros`, `ones`, `eye` slúžia na rýchle, pohodlné vytváranie špeciálnych matíc. Pomocou funkcie `mat` a textového reťazca môžete vytvárať matice v štýle MATLAB-u, teda riadky oddelené bodkočiarkou a prvky v riadku čiarkami či medzerami. Funkcie `rand`, `eye` sa vyvolajú jednoducho, napr. `rand(2,3)` bude matica náhodných čísel typu 2×3.

Funkcie `zeros`, `ones` majú nejako viac zátvoriek. Je to preto, lebo môžeme mať nuly či jedničky celočíselné, reálne alebo komplexné. Keď zadáme len usporiadanú dvojicu, napr. `zeros((2,4))` vytvorí sa celočíselná matica (vonkajšie zátvorky sú volanie funkcie `zeros`, vnútorné zas

Podobne sa určia vzdialenosti d_2, d_3, d_4 od ostávajúcich troch strán.

Výpočty bude za nás robiť Pylab, v súbore `strelci.py`, ktorý si postupne budete vytvárať. Hlavička funkcie na výpočet vzdialenosti bodu P od priamky danej dvoma bodmi A, B nech je

```
def vzdial(A,B,P):
```

a na výpočet vzdialenosti $d(x, y)$ zo vzorca (4.2), ak na vstupe zadáme vrcholy štvoruholníka a nejakého jeho vnútorného bodu P napíšte funkciu s hlavičkou

```
def optimald(P,A,B,C,D):
```

Veríme, že po predchádzajúcich skúsenostiach vám napísanie týchto dvoch funkcií nebude robiť ťažkosti. Pretože v Pylabe máme prostriedky len na minimalizáciu, musí funkcia `optimald` vrátiť $-d(x, y)$ – minimalizácia záporne vzatej funkcie je ekvivalentná s maximalizáciou pôvodnej funkcie (tak to potrebujeme my).

V module `scipy.optimize` máme veľa funkcií na hľadanie minima funkcie jednej alebo viac premenných, ale môžeme použiť len tie, ktoré nepotrebujú výpočet gradientu (parciálnych derivácií), lebo to je pre našu funkciu zložité. Hodia sa nám napr. funkcie `fmin`, `fmin_powell`. Pre konkrétny štvoruholník s vrcholmi:

$$A = (-3, -2);\ B = (4.5, -4.5);\ C = (5, 4.5);\ D = (-2, 2)$$

by sme postupovali v interaktívnom prostredí takto (predpokladáme, že ste príkazom `run strelci` načítali vami vytvorené funkcie `vzdial`, `optimald`:

```
from scipy.optimize import fmin, fmin_powell

A,B,C,D=(-3,-2),(4.5,-4.5),(5,4.5),(-2,2)
fmin(optimald,(1.0,0.0),args=(A,B,C,D),xtol=1.0e-8)
```

Výstup z `fmin`, ktorý sa vypíše na obrazovku, je optimálna poloha, t. j. bod $P = (1.535739, -0.1370124)$, pričom najmenšia vzdialenosť od strán štvoruholníka je 3.201712.

Cez nápovedu `fmin?` by ste zistili, že prvý argument je funkcia, ktorú máme minimalizovať, druhý je počiatočné priblíženie bodu, kde sa nadobúda minimum. Zaujímavý je pomenovaný argument `args` – hovorí, aké ďalšie argumenty (okrem prvého, čo sú premenné, v našom prípade súradnice bodu P v štvoruholníku) treba zadať do funkcie, ktorú optimalizujeme. Podobným štýlom sa dajú zadávať ďalšie parametre (argumenty) tiež pri numerickom integrovaní a pri riešení diferenciálnych rovníc, čo má veľký praktický význam.

Nikdy nezabúdajte na to, že práca s počítačom je **tvorivý dialóg** medzi vami a ním. Kedykoľvek vám niečo zahlási na obrazovku, venujte tomu pozornosť (hlavne, keď je tam slovko `"Error"` a keď sa to vypisuje červeno). Počítač si nevymýšľa a nesimuluje. Pokojne si prečítajte, čo vám píše (znova tá čertovská angličtina, ale musíte to skôr-neskôr prekonať :-), porozmýšľajte a snažte sa jeho správanie pochopiť. Potom sa môžete (niekedy je to aj na viackrát) pokúšať odstrániť to, čo mu na vašom „výpočte“ vadí. Ignorovaním chybových hlásení a vytrvalým opakovaním tých istých chybných príkazov (alebo ešte horšie, náhodnými experimentami a zmenami programu bez toho, aby ste rozumeli, čo vlastne robíte) sa k ničomu rozumnému nedopracujete. To je „hluboké nedorozumění “ a nie dialóg.

Poznámka: Systém Pylab sa neustále vylepšuje, jeho zdrojové kódy sú voľne prístupné na Internete. Ak si ho budete chcieť inštalovať doma, je dôležité, aby ste mali najnovšie stabilné verzie modulov, z ktorych sa skladá. Inak sa vám ľahko môže stať, že to, čo nám funguje bez problémov v škole, vám doma chodiť nebude. Uvádzame najnovšie verzie (stav z 8. septembra 2006):

Program	Verzia	URL na download	Dokumentácia
Python	2.4.3	*http://www.python.org*	(VAN ROSSUM, 2006)
IPython	0.7.2	*http://ipython.scipy.org*	(PÉREZ, 2006)
Numpy	1.0.b5	*http://www.numpy.org*	(OLIPHANT, 2005)
SciPy	0.5.1	*http://scipy.org*	(OLIPHANT, 2004)
Matplotlib	0.87.5	*http://matplotlib.sf.net*	(HUNTER, 2006)

Namiesto modulu *Numpy* (OLIPHANT, 2005) sa predtým používali moduly *Numeric* alebo *Numarray*, pre nové inštalácie ich však neodporúčame, lebo sa ďalej nevyvíjajú. Samozrejme, ak budete potrebovať usmernenie či radu pri inštalácii, radi pomôžeme (vyššieuvedená emailová adresa autora). No obyčajne stačí prečítať si príslušné súbory `README` či `INSTALL` a riadiť sa podľa nich. Pritom budete mať oveľa menej problémov s inštaláciou a udržiavaním pod operačnými systémami UNIX-áckeho typu (napr. Linux, FreeBSD, Open BSD), než keby ste používali MS Windows. Verte, drvivú väčšinu problémov, ktoré sa objavujú na elektronických konferenciách, hlásia užívatelia MS Windows. Pre výpočtovo náročné aplikácie sú OS UNIX-áckeho typu jednoznačne vhodnejšie a spoľahlivejšie.

1.2 Zadávanie vektorov, matíc a operácie s nimi

Vektory a matice sú základnými objektami v Pylabe a tvoria aj základ numerických algoritmov. Hlavnou výhodou Pylabu je, že dokáže vyko-

5 Interaktívna práca s grafickým oknom

5.1 Ukážka interaktívneho zadávania dátových bodov

Občas sa hodí, aby sme napr. mohli zadať dátové body pomocou kurzoru myši interaktívne v grafickom okne. Teda, potrebujeme, aby Pylab reagoval podľa našich pokynov na rôzne udalosti (stlačenie a uvoľnenie tlačítka myši, písanie na klávesnici, zmena veľkosti grafického okna, atď.). V Pylabe máme tieto udalosti pomenované takto

`resize_event`	zmena veľkosti grafického okna,
`draw_event`	prekreslenie obrázku,
`key_press_event`	stlačenie klávesu,
`key_release_event`	uvoľnenie klávesu,
`button_press_event`	stlačenie tlačítka myši,
`button_release_event`	uvoľnenie tlačítka myši.

Pylab používa metódu registrácie udalostí, teda povieme mu napr., že pri stlačení tlačítka myši (registrovaná udalosť, *event*) v grafickom okne má volať nami definovanú funkciu (*callback function*). Ukážeme si to na jednoduchom príklade.

Príklad 5.1 *Urobme grafické okno, kde rozsah na osi x bude* $0 \leqq x \leqq 10$ *a nech* $-6 \leqq y \leqq 6$. *Nechajme užívateľa zadávať dátové body pravým tlačidlom myši v tomto okne, ale vždy len tak, aby nasledujúci bod mal x-ovú súradnicu väčšiu, ako predchádzajúci. Zadávanie bodov sa ukončí pravým tlačidlom myši a body sa uložia v textovom formáte do súboru s názvom* `Body.txt`.

Môžeme postupovať napr. takto (nižšieuvedené príkazy nech sú uložené v súbore `bodyxy.py`):

```
from pylab import *

# Globalne premenne
V=[]        # body -- zoznam dvojic (x,y)
vc=None     # registracia funkcie pre reakciu na mys
xmax=-1     # max. hodnota x-ovej suradnice

def Points_input():
    global vc, xmax, V
    xmax=-1; V=[]
    axis([0,1,0,1])
    ax=axes()
    ax._autoscaleon=False
```

ným spôsobom, ale kliknutím myšou na značku × na lište príkazového okna, príkazy sa do histórie nezapíšu, tak si na to dajte pozor!

Po príkazoch z histórie sa môžete v Pylabe pohybovať šípkami (hore, dole). Ak viete, že chcete vykonať príkaz, ktorý je v histórii a začínal napr. `y=`, tak to napíšete a stlačíte `Ctrl-P`. Príkaz sa doplní a príp. ak ich bolo viac, môžete sa opätovným `Ctrl-P` po týchto príkazoch pohybovať smerom „dozadu“, alebo stlačením `Ctrl-N` smerom „dopredu“.

Pri práci s Pylabom sa nemôžete „stratiť“, (pravda, ak trocha viete anglicky), pretože príkazom

`objekt?` alebo `?objekt`

dostanete informácie o príslušnom objekte. To je dobre vedieť hneď na začiatku a často je to rýchlejšia alternatíva ako listovanie v dokumentácii. Skúste si napr. `plot?`, `linspace?`, `roots?`. Ak je nápoveda na viac obrazoviek, zobrazí sa pomocou špeciálneho programu – stránkovača (poznáte to podľa dvoch bodiek na samostatnom poslednom riadku v okne), vtedy prezeranie ukončíte stlačením klávesu `q`.

Jediný problém je, že na začiatku asi nebudete vedieť, na čo sa pýtať. Preto na webovej stránke `http://frcatel.fri.utc.sk/pylab.html` nájdete názvy niektorých užitočných príkazov, spolu s ich stručným vysvetlením. Prehľad príkazov nájdete aj ako prílohu tejto príručky. Potom si môžete podrobnosti doplniť vyvolaním nápovedy.

Hoci Python, náš pracovný kôň (workhorse :-) v pozadí, pracuje s mnohými typmi objektov (celé, reálne i komplexné čísla, zoznamy, asociatívne polia, usporiadané n-tice, textové reťazce), nás budú zaujímať hlavne číselné obdĺžnikové matice a vektory (ich prvky môžu byť celočíselné, reálne, ale i komplexné). Matice a vektory sú základnym dátovým typom v Pylabe a existuje veľa funkcií, ktoré sa dajú na ne aplikovať „po prvkoch“. Je teda treba mať na pamäti, že napr. **príkaz `sin(A)` neurobí len sínus jedného čísla, ale aplikuje funkciu sínus na všetky prvky matice** `A` bez toho, že by sme museli programovať cykly po prvkoch matice.

Preto programy v Pylabe vychádzajú oveľa menšie a zrozumiteľnejšie, než v tradičných programovacích jazykoch (PASCAL, FORTRAN, C/C++). Je to dané aj tým, že netreba deklarovať typy premenných; podľa toho, ako vznikli, je jasné aj akého sú typu (napr. `a=1` je celočíselná premenná, `b="facina"` je textová premenná, `c=[1,2,'tri']` je zoznam, . . .). Skrátka, všetko, čo vzniká má kdesi zaarchivovaný „rodný list“ a to stačí. Ďalšia príjemná vec je, že sa nemusíme starať o alokáciu a uvoľňovanie pamäte, čo iste ocenia hlavne C-čkári :-).

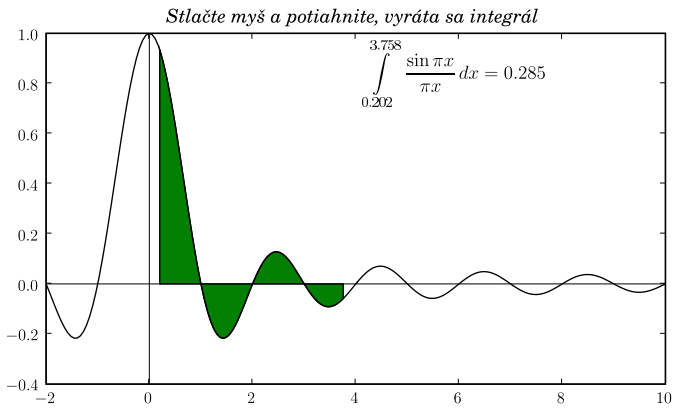
Keďže Pylab má vlastný programovací jazyk a je nim Python, môžeme pohodlne pridávať k už zabudovaným funkciám svoje vlastné funkcie,

príkladu bude čitateľ schopný vytvárať si jednoduchú interaktívnu grafiku v Pylabe.

5.2 Ukážka použitia grafického užívateľského rozhrania

MATLAB dáva možnosť vytvárať jednoduché užívateľské rozhrania. V Pylabe, vďaka modulu Matplotlib, máme k dispozícii tiež niekoľko jednoduchých prvkov na tvorbu užívateľských rozhraní, napr. grafické tlačidlá, posuvníky, nástroje na výber obdĺžnikových oblastí. Uvedieme malý príklad, ktorý naznačuje široké možnosti použitia Pylabu pri interaktívnej výučbe matematickej analýzy.

Príklad 5.2 *Vytvorme grafické okno, v ktorom bude mať užívateľ možnosť zadávania hraníc a, b integrálu $\int_a^b \frac{\sin \pi x}{\pi x}\,\mathrm{d}x$ pomocou myši. Po zadaní hraníc sa v grafickom okne zobrazí výsledok, napr. ako na nasledujúcom obrázku.*



Na výber hraníc integrálu použijeme grafický prvok `SpanSelector`, ktorým sa dajú v grafickom okne vyberať zvislé alebo vodorovné pásy. Nasledujúce príkazy budeme zapisovať do súboru `definteg.py`.

Najskôr ošetríme potrebné importy (grafika, numerické integrovanie). Potom pomocou lambda notácie (je vysvetlená na str. 47) zavedieme funkciu `fcn`, ktorú budeme integrovať.

```
from pylab import *
from matplotlib.widgets import SpanSelector
from scipy.integrate import quad

fcn=lambda x: sin(pi*x)/(pi*x)
```

Ďalšia prekážka, ktorú musíte prekonať, je zvyknúť si na iný, hoci pohodlnejší štýl práce v Pylabe, objavovať a naučiť sa využívať jeho bohaté možnosti. Nie je to PASCAL, ani C-čko, takže sa ho nesnažte podľa vzoru týchto jazykov komplikovať a zneprehľadňovať. Jasný, prehľadný a logický program je ako umelecké dielo. Nič tam nesmie byť zbytočné a všetky časti musia byť veľmi dobre vybrané a zosúladené. Potom môžete mať radosť z toho, že ste niečo také dokázali vytvoriť.

Ďakujem kolegom a recenzentom Janovi Bušovi a Ladislavovi Ševčovičovi za možnosť vydania tejto učebnice v rámci série príručiek k Open Source softvéru,[2] za dôkladné testovanie Pylabu a cenné pripomienky k samotnému systému ako aj k tejto príručke. Ďakujem tiež kolegyni Alžbete Klaudiny za veľmi dôkladné prečítanie tohoto textu a početné navrhnuté opravy, ktoré pomohli zlepšiť jeho kvalitu.

Žilina 14. septembra 2006 Michal Kaukič

[2]Elektronické verzie tejto a ďalších príručiek vytvorených v rámci projektu KEGA sú prístupné v priečinkoch na adrese `http://people.tuke.sk/jan.busa/kega`.


```
Drawn=None       # ci uz bola nakreslena vyplnena plocha

def onselect(vmin, vmax):
    global Drawn
    if Drawn:      # vycisti vyplnenu plochu aj vypisy
                   # o hraniciach a hodnote integralu
        ax.patches,ax.texts=[],[]
    else:
        Drawn=True

    xx=linspace(vmin,vmax, int(200*(vmax-vmin)/12.0))
    yy=sinc(xx)
    xx=array([vmin]+list(xx)+[vmax,vmin]) # uzavrety polygon
    yy=array([0.0]+list(yy)+[0.0,0.0])    # pre vyplnenie
    fill(xx,yy,facecolor='g')
    text (6,0.7,
          r"$\displaystyle{\int\limits_{%2.3f}^{%2.3f}\
          {\frac{\sin \uppi x}{\uppi x}}\,dx=%2.3f}$" \
          %(vmin,vmax,quad(sinc,vmin,vmax)[0]),
          horizontalalignment='center',fontsize=14)
    draw()                    # aby sa aktualizoval obrazok

span = SpanSelector(ax, onselect, 'horizontal')
show()                        # prve nakreslenie obrazka
```

Okrem definície obslužnej funkcie, na konci súboru je do aktuálneho obrázka pridaný grafický prvok `SpanSelector`, ktorému je priradená táto obslužná funkcia `onselect`.

Iné príklady použitia grafických prvkov nájdete na domovskej stránke Matplotlibu, *http://matplotlib.sourceforge.net*, je tam v hlavnom menu položka *Examples (zip)*, odtiaľ si ich môžete stiahnuť. Myslíme si, že prezeranie zdrojových textov týchto príkladov a ich modifikácia je najlepší spôsob, ako sa zoznámiť s Pylabom a jeho grafikou.

Úvod

Prvoradým cieľom tejto učebnice je poskytnúť základné informácie, potrebné k úspešnej práci s programovým systémom Pylab a s jazykom Python, na ktorom je tento systém založený. Autor tohoto textu využíva Pylab na Fakulte riadenia a informatiky Žilinskej univerzity pri výučbe predmetov *Numerické metódy, UNIX – vývojové prostredie, Moderné približné metódy, Softvérové nástroje pre inžinierov (Open Source)*. Postupne by sme chceli tento systém zaviesť aj do výučby ďalších matematických a príbuzných predmetov a boli by sme radi, keby sa začal používať aj na iných slovenských vysokých a stredných školách. Príklady, uvedené v tomto texte, sú hlavne z oblasti numerickej analýzy, vrátane rozsiahlejších ukážok použitia Pylabu na problémy interpolácie a aproximácie dát, numerickej integrácie a riešenia nelineárnych rovníc a ich sústav.

Sme presvedčení, že zvládnutie tohto elegantného, jasného a ľahko naučiteľného programovacieho prostriedku sa čitateľovi vyplatí aj do budúcnosti. Je to mohutný systém s veľkými výpočtovými i grafickými možnosťami, určený na prácu aj s veľkými objemami dát a s ich vizualizáciou. V pozadí je kvalitný univerzálny programovací jazyk Python (VAN ROSSUM, 2006), ktorý máte v Pylabe plne k dispozícii. V prípade potreby si môžete do systému pridať ďalšie moduly (napr. pre prácu s databázami SQL, lineárne programovanie, prezentáciu výpočtov a dát na webe a pod.).

Nemusíte však veľa vedieť o programovaní v Pythone, pretože Pylab je jazyk veľmi vysokej úrovne, inšpirovaný známym komerčným systémom MATLAB[1]. Pri práci v ňom dosiahnete s minimálnou „programátorskou“ námahou veľké výsledky a môžete sa viac venovať skutočne zaujímavým a podstatným veciam nielen v matematicky zameraných predmetoch, ale aj vo svojej ďalšej odbornej práci. Zo skúseností vieme a veríme, že mnohým z vás Pylab a Python pomôže napr. pri spracovaní semestrálok, bakalárskych či diplomových prác a aj neskôr, keď budete v zamestnaní.

Výhodou Pylabu je vysoký stupeň kompatibility s MATLAB-om, ktorý je veľmi rozšírený hlavne vo vzdelávacích inštitúciách, ale aj v priemysle, vďaka existencii rôznych rozšírení (toolboxov) hlavne pre oblasť inžinierskych výpočtov. To, čo MATLAB nemá, je univerzálny programovací jazyk (u nás Python), široko využiteľný aj samostatne. Ak však potrebujete používať už hotové programy, napísané pre MATLAB, odporúčame vám Open Source systém Octave (BUŠA, 2006).

[1] MATLAB je registrovaná obchodná značka softvérovej firmy The MathWorks,Inc.

Príloha – Zoznam najpoužívanejších funkcií

Tabuľka 2: Najpoužívanejšie funkcie pre grafiku

`axis`	nastaví alebo vráti aktuálne hranice na súrad. osiach
`cla`	vyčistí aktuálny súradnicový systém
`clf`	vyčistí celé obrázkové okno, bude bez súradníc
`close`	zatvorí (aktuálne) obrázkové okno
`colorbar`	pridá farebnú stupnicu do aktuálneho obrázku
`contour`	urobí vrstevnicový graf
`contourf`	vrstevnicový graf, ale farebne vyplnený medzi vrstevnicami
`draw`	prinúti aktuálny obrázok, aby sa prekreslil
`figure`	vytvorí alebo zmení aktívny obrázok
`fill`	kreslenie (vyplnených) mnohouholníkov
`gca`	vráti aktuálny súrad. systém (pre modifikáciu vlastností)
`gcf`	vráti aktuálny obrázok (pre modifikáciu vlastností)
`grid`	prepína zobrazenie sieťky na grafe
`hold`	určuje, či sa grafické objekty pridávajú, alebo sa zakaždým mažú
`legend`	legenda pre aktuálne osi
`plot`	urobí normálny, čiarový graf – ASI NAJPOUŽÍVANEJŠÍ PRÍKAZ
`pcolor`	pre funkciu dvoch premenných, hodnoty znázornené farbami
`polar`	graf v polárnych súradniciach
`savefig`	uloženie aktuálneho obrázka (`.jpg`, `.png`, `.eps`)
`show`	ukázať obrázky (v neinteraktívnom režime)
`subplot`	urobí subplot (`poc_riadkov,poc_stlpcov,akt_sursys`)
`text`	pridá text na pozíciu (x, y) v aktuálnom súradnicovom systéme
`title`	pridá titulok k aktuálnemu súradnicovému systému
`xlabel`	nadpis pre x-ovú os
`ylabel`	nadpis pre y-ovú os

Táto publikácia vznikla s prispením grantovej agentúry SR KEGA v tematickej oblasti „Nové technológie vo výučbe“– projekt: 3/2158/04 – „Využitie Open Source softvéru vo výučbe na vysokých školách“.

Recenzovali: Ján Buša
Ladislav Ševčovič

ISBN 80-8073-634-0

Sadzba programom pdfTEX

Literatúra

Buša, J. 2006. *Octave, Rozšírený úvod*, 105 s. 4, 19

van Rossum, G. 2006. *Python Documentation*, online dokumentácia na stránke `http://www.python.org/doc/`. 4, 10

Pérez, F. 2006. *IPython. An enhanced Interactive Python*, online na stránke `http://ipython.scipy.org/doc/manual/`. 10

Oliphant, T. E. 2005. *Guide to NumPy*, 247 s. 10

Oliphant, T. E. 2004. *SciPy Tutorial*, 42 s. 10, 56

Kaukič, M. 1998. *Numerická analýza I. Základné problémy a metódy*. Žilina, MC Energy s. r. o., 202 s. 19

Hunter, J. 2006. *The Matplotlib User's Guide*, 79 s. 10, 28

Venables W. N., Smith D. H. and the R Development Core Team. 2006. *An Introduction to R*, 99 s. 56

Joyner D., Stein W. 2006. *SAGE Tutorial*, 98 s. 56