MATLAB†	SCILAB‡

`% definiowanie funkcji`
`function [dx]=VDerPol(t,y)`
`global c;`
`dx=[y(2);`
`  c*(1-y(1)^2)*y(2)-y(1)];`
	`// definiowanie układu`
	`function [f]=VDerPol(t,y,c)`
	`f(1)=y(2);`
	`f(2)=c*(1-y(1)^2)*y(2)-y(1);`
	`endfunction`

† W Matlabie funkcja VDerPol() jest zdefiniowana w osobnym pliku dyskowym w katalogu work.

‡ W Scilabie funkcja VDerPol() może, ale nie musi być zdefiniowana w osobnym pliku dyskowym. Jeśli stanowi część programu głównego, to rezygnujemy z wywoływania jej poleceniem `getf()`.

`% warunki początkowe`
`c=0.4; m=500;`
`y0=[-2.5;2.5];`
`global  c;`
	`//warunki początkowe`
	`m=500; T=30; c=0.4;`
	`t=linspace(0,T,m);`
	`y0=[-2.5;2.5];`

`% rozwiązywanie`	`// rozwiązywanie` `getf('VDerPol.sci');`
`[T,y]=ode45('VDerPol',0,m,y0);`	`[y]=ode(y0,0,t,VDerPol);`
`% kreślenie rozwiązania` `plot(y(:,1),y(:,2))`	`// kreślenie rozwiązania` `plot2d(y(1,:),y(2,:))`

† Proszę zwrócić na sposób wykorzystania zmiennej globalnej

‡ Więcej o poleceniach `plot()` i `plot2d()` w podrozdziałach 5.1 i 5.2.

4.3.4. Optymalizacja

Zagadnienie optymalizacji pojawia się w wielu problemach praktycznych, gdzie poszukujemy wartości parametrów, dla których pewne funkcje opisujące analizowany problem osiągają maksimum lub minimum. Rozwiązanie zagadnienia optymalizacyjnego polega więc na znalezieniu najlepszego, względem ustalonego kryterium, rozwiązania należącego do zbioru rozwiązań dopuszczalnych. Zagadnienia optymalizacji, nazywane również zadaniem programowania, są w Matlabie i Scilabie bardzo szeroko opracowane. Niestety w Matlabie stanowią część osobnej biblioteki *Optimization Toolbox*, którą należy dodatkowo dokupić,

bez jej obecności niektóre funkcje opisywane w tym rozdziale mogą być niedostępne. Ograniczenia tego typu nie występują w Scilabie.

Programowanie liniowe

W zagadnieniu programowania liniowego funkcja celu jest liniową funkcją wielu zmiennych. Zagadnienie programowania możemy zapisać w postaci:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{c}^T\mathbf{x} \\ &\mathbf{A}\mathbf{x} \leq \mathbf{b} \end{aligned} \tag{4.3.8}$$

gdzie: **A** – macierz współczynników ograniczeń,
b – wektor kolumnowy współczynników ograniczeń,
c – wektor kolumnowy współczynników funkcji celu,
x – $[x_1,x_2,\ldots,x_n]$ wektor rozwiązań.

Tytułem przykładu rozwiążemy następujący problem:

$$\min_{\mathbf{x}} \left\{-x_1 - 1.4x_2\right\}, \tag{4.3.9}$$

dla ograniczeń nierównościowych:

$$\begin{array}{rcrcl} x_1 & + & x_2 & \leq & 400, \\ x_1 & + & 2x_2 & \leq & 580, \\ x_1 & & & \leq & 300, \\ -x_1 & & & \leq & 0, \\ -x_2 & & & \leq & 0. \end{array} \tag{4.3.10}$$

Skrypty mogą przyjąć następującą postać:

MATLAB	SCILAB
`c=[-1 -1.4]';`	`c=[-1 -1.4]';`
`A=[1 1; 1 2; 1 0; -1 0; 0 -1];`	`A=[1 1; 1 2; 1 0; -1 0; 0 -1];`
`b=[400 580 300 0 0]';`	`b=[400 580 300 0 0]';`
`[x,lg]=linprog(c,A,b);`	`[x,lg]=linpro(c,A,b);`

x - wektor rozwiązań, lg - współczynniki Lagrange'a.

Programowanie kwadratowe

Do rozwiązywania zagadnień programowania kwadratowego można w zasadzie wykorzystać metody programowania nieliniowego omówione

w następnym podpunkcie. Jednak oba systemy oferują wyspecjalizowane polecenia do rozwiązywania tego typu problemów. Wymieńmy tu `quadprog()` i `quapro()` odpowiednio dla Matlaba i Scilaba. Oba polecenia rozwiązują zadanie programowania kwadratowego zapisane w postaci:

$$\begin{aligned} \min_{\mathbf{x}} \frac{1}{2}\mathbf{x}\mathbf{H}^T\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \mathbf{A}\mathbf{x} \leq \mathbf{b} \end{aligned} \tag{4.3.11}$$

gdzie: **A** – macierz współczynników ograniczeń,
b – wektor kolumnowy współczynników ograniczeń,
c – wektor kolumnowy współczynników funkcji celu,
H – macierz współczynników funkcji celu,
x – $[x_1,x_2,\ldots,x_n]$ wektor rozwiązań.

Sposób wykorzystania obu poleceń pokażemy, poszukując minimum następującej funkcji celu:

$$\min_{\mathbf{x}} \left\{(x_1-3)^2 + (x_2-4)^2\right\}, \tag{4.3.12}$$

dla ograniczeń nierównościowych:

$$\begin{array}{rcrcl} x_1 & + & 2x_2 & \leq & 3, \\ 3x_1 & + & x_2 & \leq & 4, \\ & & \mathbf{x} & \geq & 0, \end{array} \tag{4.3.13}$$

gdzie występujące w równaniu (4.3.11) **H** i **c** przyjmują postać:

$$H = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \quad c = \begin{pmatrix} -6 \\ -8 \end{pmatrix}.$$

MATLAB	SCILAB
`H=[2 0; 0 2];`	`H=[2 0; 0 2];`
`c=[-6; -8];`	`c=[-6; -8];`
`A=[1 2; 3 1];`	`A=[1 2; 3 1];`
`b=[3; 4];`	`b=[3; 4];`
`[x]=quadprog(H,c,A,b);`	`[x]=quapro(H,c,A,b);`

x - wektor rozwiązań.

Programowanie nieliniowe

Prosty przykład optymalizacji funkcji nieliniowej pokazano w punkcie 4.2.2. Poniżej zostanie pokazany inny, nieco bardziej złożony przykład optymalizacji funkcji dwu zmiennych. Poszukujemy minimum funkcji Rosenbrocka

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

rozpiętej w przestrzeni liczb rzeczywistych $\mathbb{R}^2$. Pierwszym krokiem jest oczywiście napisanie funkcji użytkownika obliczającej wartości funkcji `f` dla argumentów x_1, x_2. W przypadku Scilaba należy również obliczyć gradient funkcji.

MATLAB	SCILAB‡

`function [f]=cost(x)`
`  f=100*(x(2)-x(1)^2)^2...`
`  +(1-x(1))^2;`
	`function [f,g,ind]=cost(x,ind)`
	`f=100*(x(2)-x(1)^2)^2+...`
	`(1-x(1))^2;`
	`g=[-400*x(1)*(x(2)-x(1)^2)...`
	`-2*(1-x(1));200*(x(2)-x(1)^2)]`
	`endfunction`

‡ Zmienna `ind` jest parametrem nie wykorzystanym, ale musi być zdefiniowana. Zmienna `g` to gradient funkcji `f` w punkcie `x`. Zadajemy go w postaci wektora kolumnowego zwierającego pochodne funkcji $f(x_1, x_2)$ liczone po x_1 i x_2, $g = [\frac{\partial f}{\partial x_1}; \frac{\partial f}{\partial x_2}]$.

Samo poszukiwanie minimum funkcji to wydanie polecenia `fminsearch()` dla Matlaba lub `optim()` dla Scilaba.

MATLAB†	SCILAB†

`[xopt,fval]= ...`
`  fminsearch(@cost,[-1.2,1])`
`xopt= 1.0  1.0`
`fval = 8.177e-10`
	`[f,xopt]=...`
	`  optim(cost,[-1.2;1])`
	`f = 8,177e-10`
	`xopt=! 1; 1 !`

† Wektor `[-1.2;1]` to warunki początkowe.

W tablicy 4.5 zestawiono wybrane funkcje rozwiązujące zagadnienia optymalizacji.

Tablica 4.5. Polecenia umożliwiające rozwiązywanie zagadnień optymalizacji.

MATLAB†	SCILAB	Opis
`linprog()`	`linpro()`	programowanie liniowe
`quadprog()`	`quapro()`	programowanie kwadratowe
`leastsq()`	`leastsq()`	metoda najmniejszych kwadratów
`fminbnd()`	`optim()`	programowanie nieliniowe
`fminsearch()`	`optim()`	programowanie nieliniowe

† Polecenia dostępne po zakupieniu biblioteki *Optimization Toolbox*.

4.4. Skrypty

Skrypt to zbiór poleceń, komentarzy, wywołań funkcji, zapisany w odrębnym pliku dyskowym, realizujący określony przez użytkownika algorytm. Praca w systemach Matlab i Scilab powinna polegać na pisaniu skryptów, gdyż tylko wtedy możemy w pełni kontrolować proces obliczeń i maksymalnie wykorzystać możliwości systemu.

4.5. Sztuka programowania

4.5.1. Rady dla początkujących

Jeśli uznamy programowanie, może nieco na wyrost, za formę sztuki, możemy stwierdzić, że nie jest możliwe jej nauczenie. Można jednak pokusić się o podanie pewnych wskazówek dla jej adeptów, które mogą ułatwić pisanie programów krótkich, optymalnych i szybkich, realizujących bezbłędnie postawione im zadania.

Kilka prostych zasad

— Zanim zabierzemy się do pisania programu komputerowego, analizowany problem należy rozwiązać w sposób klasyczny na kartce papieru, oczywiście w takim zakresie, w jakim jest to możliwe. Pominięcie tego kroku jest gwarancją klęski.
— Każdy program musi mieć jasno określone parametry wejściowe i wyjściowe.

— Nazwy zmiennych w programie powinny odzwierciedlać ich role w algorytmie.

4.5.2. Pisanie programów wydajnych

Dobrze napisany program jest wynikiem spełnienia dwóch warunków:
— dobrego zrozumienia wybranego algorytmu,
— biegłej znajomości używanego narzędzia.

Znaczenie algorytmu

Rozwiązanie problemu jest niejednokrotnie utożsamiane ze znalezieniem właściwego algorytmu. Głównym niebezpieczeństwem, na jakie napotykamy, jest skłonność do bezkrytycznego wykorzystywania znalezionego algorytmu w swoim programie. Jako ilustracje rozważmy dwa zagadnienia. Pierwsze to zadanie wyznaczania liczb Fibonacciego[5]. Definicja liczby Fibonacciego ma charakter rekurencyjny i jest następująca:

$$F_k = \begin{cases} 1, & \text{k=1,2}, \\ F_{k-1} + F_{k-2}, & \text{k}\geq 3. \end{cases} \tag{4.5.1}$$

Bezpośrednia implementacja wzoru (4.5.1) w Scilabie prowadzi do następującej definicji funkcji:

```
function r=fib(n)
 if n <=2 then
   r=1;
 else
   r=fib(n-1)+fib(n-2);
 end
endfunction // tylko dla Scilaba
```

Wyznaczenie piątej z kolei, licząc od 1, liczby Fibonacciego wymaga dziewięciu wywołań funkcji `fib()`, jest to dużo i wydaje się, że procedura jest bardzo „czasochłonna". Można ją uprościć poprzez uproszczenie można uzyskać drogą uproszczenia podwójnego wywołania rekurencyjnego

[5] Leonardo z Pizy Fibonacci (1170-1240), włoski matematyk, propagator arabskiego (dziesiętnego) sposobu liczenia, główne dzieła poświęcił arytmetyce, algebrze i geometrii.