POČÍTAČOVÁ PODPORA V ELEKTROTECHNICE

Ing. Lenka Šroubová, Ph.D.
lsroubov@kte.zcu.cz

Ing. Petr Kropík, Ph.D.
pkropik@kte.zcu.cz

Katedra teoretické elektrotechniky
Fakulta elektrotechnická
Západočeská univerzita v Plzni

Místnost: EK602

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Práce s maticemi a vektory

Přidání řádku k matici:

```
D=[1:5;9,3,4,3,2;1:2:9;9:-1:5]

D =
    1    2    3    4    5
    9    3    4    3    2
    1    3    5    7    9
    9    8    7    6    5
```

```
1:4
ans =
     1   2   3   4
```

`D(5,:)=[1:4]` – přidání 5. řádku

nelze: řádek musí mít 5 sloupců (prvků) jako matice

```
size(D)
ans =
     4   5
```

Práce s maticemi a vektory

Přidání řádku k matici:

```
D=[1:5;9,3,4,3,2;1:2:9;9:-1:5]
D =
    1    2    3    4    5
    9    3    4    3    2
    1    3    5    7    9
    9    8    7    6    5
```

```
1:5
ans =
     1   2   3   4   5
```

`D(5,:)=[1:5]` – přidání 5. řádku

```
D =
    1    2    3    4    5
    9    3    4    3    2
    1    3    5    7    9
    9    8    7    6    5
    1    2    3    4    5
```

```
size(D)
ans =
     5   5
```

Práce s maticemi a vektory

Nahrazení prvku v matici:

```
D =
     1     2     3     4     5
     9     3     4     3     2
     1     3     5     7     9
→    9     8     7     6     5
     1     2     3     4     5
```

```
D(4,2) = 10
```

– nahrazení prvku (čísla 8) na 4. řádku, ve 2. sloupci číslem 10

```
D =
     1     2     3     4     5
     9     3     4     3     2
     1     3     5     7     9
→    9    10     7     6     5
     1     2     3     4     5
```

Práce s maticemi a vektory

Nahrazení prvku v matici:

```
D =
     1     2     3     4     5
     9     3     4     3     2
     1     3     5     7     9
     9    10     7     6     5
     1     2     3     4     5
```

```
D(3:5, 1:3) = 0.001
```

– nahrazení prvků ve **3. až 5. řádku** a **1. až 3. sloupci** číslem 0.001

```
D =
    1.000    2.000    3.000    4.000    5.000
    9.000    3.000    4.000    3.000    2.000
    0.001    0.001    0.001    7.000    9.000
    0.001    0.001    0.001    6.000    5.000
    0.001    0.001    0.001    4.000    5.000
```

Práce s maticemi a vektory

Zrušení sloupce nebo řádku:

```
D =
    1.000    2.000    3.000    4.000    5.000
    9.000    3.000    4.000    3.000    2.000
    0.001    0.001    0.001    7.000    9.000
    0.001    0.001    0.001    6.000    5.000
    0.001    0.001    0.001    4.000    5.000
```

```
D(:,4) = []
```

– zrušení 4. sloupce – vymažeme jej přiřazením prázdné matice `[]`

```
D =
    1.000    2.000    3.000    5.000
    9.000    3.000    4.000    2.000
    0.001    0.001    0.001    9.000
    0.001    0.001    0.001    5.000
    0.001    0.001    0.001    5.000
```

```
size(D)
ans =
     5     4
```

Práce s maticemi a vektory

Přístup k prvkům matice :

```
D =
    1.000    2.000    3.000    5.000
    9.000    3.000    4.000    2.000
    0.001    0.001    0.001    9.000
    0.001    0.001    0.001    5.000
    0.001    0.001    0.001    5.000
```

`D(:,5)` – přístup k prvkům ve **všech** řádcích a **5. sloupci**
nelze: 5. sloupec už **neexistuje**

```
size(D)
ans =
     5   4
```

– před zrušením sloupce matice `D` měla 5 řádků a 5 sloupců, 4. sloupec byl odstraněn přiřazením prázdné matice `[]`, nyní má matice `D` jen 4 sloupce.

Práce s maticemi a vektory

5. sloupec k matici opět přidáme:

```
D =
    1.000    2.000    3.000    5.000
    9.000    3.000    4.000    2.000
    0.001    0.001    0.001    9.000
    0.001    0.001    0.001    5.000
    0.001    0.001    0.001    5.000
D(:,5) = [7;9;3;2;5]
D =
    1.000    2.000    3.000    5.000    7.000
    9.000    3.000    4.000    2.000    9.000
    0.001    0.001    0.001    9.000    3.000
    0.001    0.001    0.001    5.000    2.000
    0.001    0.001    0.001    5.000    5.000
```

Měl by to být sloupcový vektor – přidáváme sloupec, proto středník ;

Stejně by se k matici přidal řádek nebo více řádků nebo více sloupců… Samozřejmě se musí dodržet v přiřazení, zda dodám sloupec nebo řádky.

Práce s maticemi a vektory

Zrušení sloupce nebo řádku:

```
D =
    1.000    2.000    3.000    5.000    7.000
    9.000    3.000    4.000    2.000    9.000
    0.001    0.001    0.001    9.000    3.000
    0.001    0.001    0.001    5.000    2.000
    0.001    0.001    0.001    5.000    5.000
```

```
D(3:5,1:3) = []
```

– zrušení **3. až 5. řádku** a **1. až 3. sloupce**

nelze: není možné „vykousnout“ kus matice

Práce s maticemi a vektory

Nahrazení řádku nebo sloupce:

```
D =
    1.000    2.000    3.000    5.000    7.000
    9.000    3.000    4.000    2.000    9.000
    0.001    0.001    0.001    9.000    3.000
    0.001    0.001    0.001    5.000    2.000
    0.001    0.001    0.001    5.000    5.000
```

```
D(:,4) = zeros(5,1)
```

Práce s maticemi a vektory

Nahrazení řádku nebo sloupce:

```
D =
    1.000    2.000    3.000    5.000    7.000
    9.000    3.000    4.000    2.000    9.000
    0.001    0.001    0.001    9.000    3.000
    0.001    0.001    0.001    5.000    2.000
    0.001    0.001    0.001    5.000    5.000
```

5 řádků, 1 sloupec => sloupcový vektor plný nul

`D(:,4) = zeros(5,1)` – nahrazení 4. sloupce nulami

```
D =
    1.000    2.000    3.000    0.000    7.000
    9.000    3.000    4.000    0.000    9.000
    0.001    0.001    0.001    0.000    3.000
    0.001    0.001    0.001    0.000    2.000
    0.001    0.001    0.001    0.000    5.000
```

Práce s maticemi a vektory

Přidání prvku :

```
D =
    1.000    2.000    3.000    0.000    7.000
    9.000    3.000    4.000    0.000    9.000
    0.001    0.001    0.001    0.000    3.000
    0.001    0.001    0.001    0.000    2.000
    0.001    0.001    0.001    0.000    5.000
D(7,6) = -3
```

Práce s maticemi a vektory

Přidání prvku :

```
D =
    1.000    2.000    3.000    0.000    7.000
    9.000    3.000    4.000    0.000    9.000
    0.001    0.001    0.001    0.000    3.000
    0.001    0.001    0.001    0.000    2.000
    0.001    0.001    0.001    0.000    5.000
```

`D(7,6) = -3` – přidání prvku na 7. řádek a 6. sloupec

```
D =
    1.000    2.000    3.000    0.000    7.000    0.000
    9.000    3.000    4.000    0.000    9.000    0.000
    0.001    0.001    0.001    0.000    3.000    0.000
    0.001    0.001    0.001    0.000    2.000    0.000
    0.001    0.001    0.001    0.000    5.000    0.000
    0.000    0.000    0.000    0.000    0.000    0.000
    0.000    0.000    0.000    0.000    0.000   -3.000
```

Toto lze provést, zbytek matice se vyplní **nulami**.

Práce s maticemi a vektory

```
D =
    1.000    2.000    3.000    0.000    7.000    0.000
    9.000    3.000    4.000    0.000    9.000    0.000
    0.001    0.001    0.001    0.000    3.000    0.000
    0.001    0.001    0.001    0.000    2.000    0.000
    0.001    0.001    0.001    0.000    5.000    0.000
    0.000    0.000    0.000    0.000    0.000    0.000
    0.000    0.000    0.000    0.000    0.000   -3.000
```

```
E = D(2:4,3:5)
E =
    4.000    0.000    9.000
    0.001    0.000    3.000
    0.001    0.000    2.000
```

```
F = E.'
F =
    4.000    0.001    0.001
    0.000    0.000    0.000
    9.000    3.000    2.000
```

```
size(D)
ans =
     7     6

size(E)
ans =
     3     3

size(F)
ans =
     3     3
```

Práce s maticemi a vektory

```
D =
    1.000    2.000    3.000    0.000    7.000    0.000
    9.000    3.000    4.000    0.000    9.000    0.000
    0.001    0.001    0.001    0.000    3.000    0.000
    0.001    0.001    0.001    0.000    2.000    0.000
    0.001    0.001    0.001    0.000    5.000    0.000
    0.000    0.000    0.000    0.000    0.000    0.000
    0.000    0.000    0.000    0.000    0.000   -3.000
```

```
E = D(2:4,3:5);
F = E.';
D(2:4,3:5) = F
```

```
D =
    1.000    2.000    3.000    0.000    7.000    0.000
    9.000    3.000    4.000    0.001    0.001    0.000
    0.001    0.001    0.000    0.000    0.000    0.000
    0.001    0.001    9.001    3.000    2.000    0.000
    0.001    0.001    0.001    0.000    5.000    0.000
    0.000    0.000    0.000    0.000    0.000    0.000
    0.000    0.000    0.000    0.000    0.000   -3.000
```

Práce s maticemi a vektory

```
D =
    1.000    2.000    3.000    0.000    7.000    0.000
    9.000    3.000    4.000    0.001    0.001    0.000
    0.001    0.001    0.000    0.000    0.000    0.000
    0.001    0.001    9.001    3.000    2.000    0.000
    0.001    0.001    0.001    0.000    5.000    0.000
    0.000    0.000    0.000    0.000    0.000    0.000
    0.000    0.000    0.000    0.000    0.000   -3.000
size(D)
ans =
     7     6
D(2:6,:) = []
```

– zrušení 2. až 6. řádku

```
D =
    1.000    2.000    3.000    0.000    7.000    0.000
    0.000    0.000    0.000    0.000    0.000   -3.000
size(D)
ans =
     2     6
```

Pozor: rozměr matice se mění!
Původní matice již neexistuje.

Práce s maticemi a vektory

Zrušení sloupců nebo řádků:

```
D =
    1.000    2.000    3.000    0.000    7.000    0.000
    0.000    0.000    0.000    0.000    0.000   -3.000
```

`D(m,:) = []`
zrušení *m*-tého řádku (a **všech** sloupců)
`D(:,n) = []`
zrušení *n*-tého sloupce (a **všech** řádků)

`D(:,:) = []`

– zrušení celé matice, **všech** řádků a **všech** sloupců, vznikne prázdná matice

```
D =
     Empty matrix
```

Práce s vektory a maticemi

`linspace(od, do, počet_prvků_mezi_od_do)` – vytvoří řádkový vektor se zadaným počtem prvků **rovnoměrně** rozložených mezi **počáteční** a **koncovou** hodnotou

Např.:

`linspace(0,10,5)` – 5 prvků mezi 0 a 10

```
ans =
         0    2.5000    5.0000    7.5000   10.0000
```

`logspace(exp_od, exp_do, počet_prvků)` – vytvoří řádkový vektor, avšak se zadaným počtem prvků **logaritmicky** rozložených mezi 10^{exp_od} a 10^{exp_do}.

Např.:

`logspace(0,1,5)` – 5 prvků mezi 10^0 a 10^1, tj. 5 bodů mezi 1 a 10

```
ans =
    1.0000    1.7783    3.1623    5.6234   10.0000
```

Práce s vektory a maticemi

```
0:2:4
ans =
     0     2     4
```

– prvky mezi 0 a 4 s krokem 2

```
linspace(0,4,3)
ans =
     0     2     4
```

– 3 prvky mezi 0 a 4 (nezadán krok)
– lineární rozložení prvků

```
logspace(0,4,3)
ans =
     1   100 10000
```

– 3 prvky mezi 10^0 a 10^4, tj. mezi 1 a 10000
– logaritmické rozložení prvků

Práce s vektory a maticemi

```
0:2:4
ans =
     0     2     4
```

– prvky mezi 0 a 4 s krokem 2

není zadán počet prvků

```
linspace(0,4,3)
ans =
     0     2     4
```

– 3 prvky mezi 0 a 4 (nezadán krok)
– lineární rozložení prvků

není zadán krok

```
logspace(0,4,3)
ans =
     1   100 10000
```

– 3 prvky mezi 10^0 a 10^4, tj. mezi 1 a 10000
– logaritmické rozložení prvků

není zadán krok

Funkce pro práci s maticemi

```
D=[1,2,3;4,5,6;7,8,-9]
D =
     1     2     3
     4     5     6
     7     8    -9
```

diag(D)

– vypíše hlavní diagonálu matice **D** (hlavní diagonála je tvořena všemi prvky D_{mn}, kde $m = n$.)

```
diag(D)
ans =
     1          prvek D11
     5          prvek D22
    -9          prvek D33
```

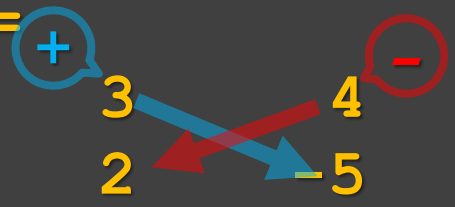
det(D)

– vypočte determinant matice **D** (jen pro **čtvercové matice**). Determinantem čtvercové matice řádu n nazýváme součet všech součinů n prvků této matice takových, že v žádném z uvedených součinů se nevyskytují dva prvky z téhož řádku ani z téhož sloupce.

```
det(D)
ans =
    54
```

Funkce pro práci s maticemi

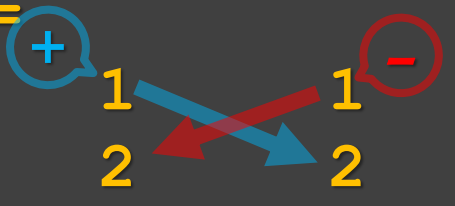
```
A=[3,4;2,-5]
A =
```



Determinant matice
det(**A**) = 3*(-5) – 4*2 = -23

```
det(A)
ans =
    -23
```

```
B=[1,1;2,2]
B =
```



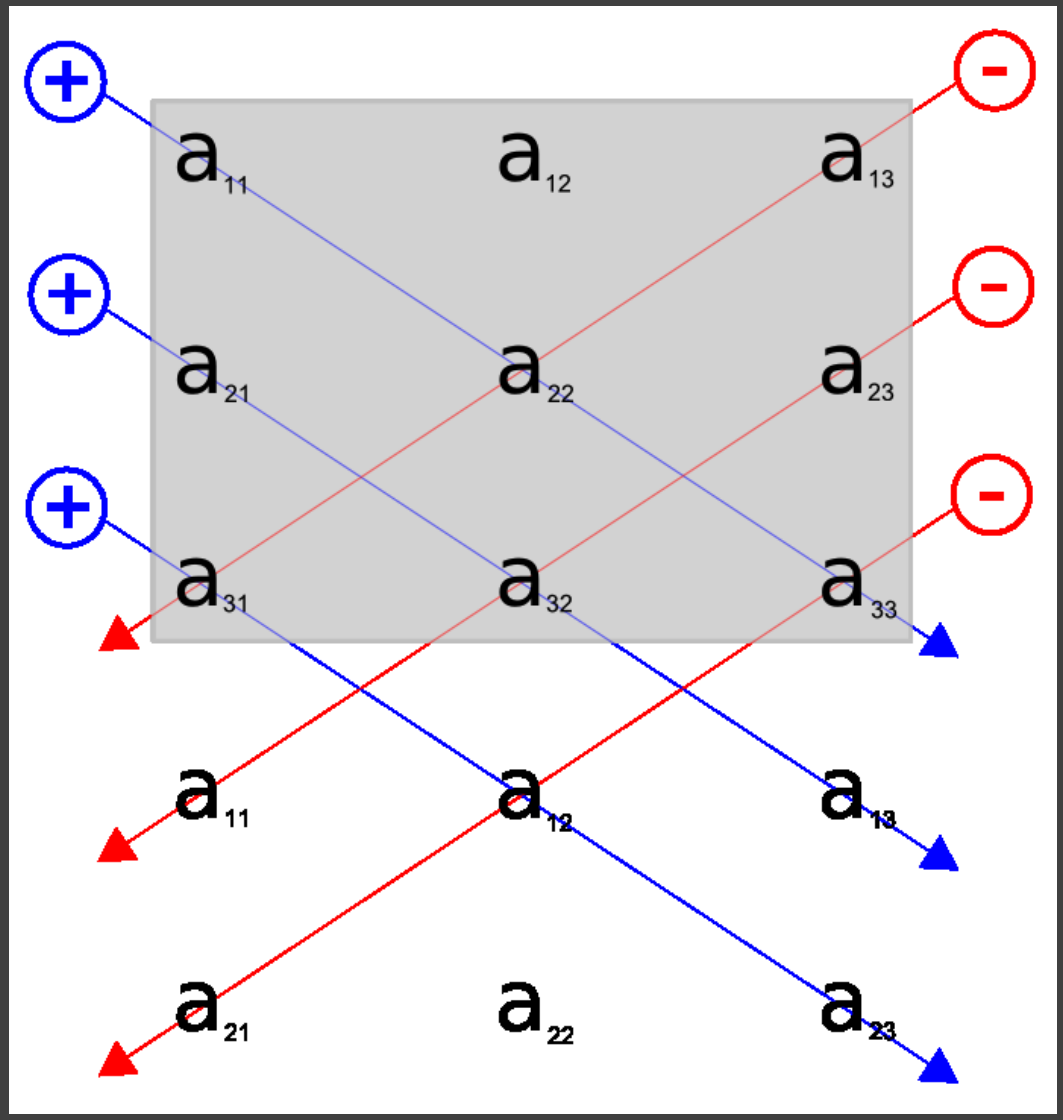
Determinant matice
det(**B**) = 1*2 – 1*2 = 0

```
det(B)
ans =
    0
```

Matice, která má determinant **rovný nule** se nazývá **singulární**.
Matice, jejíž determinant je **nenulový**, se nazývá **regulární**.
Je-li matice koeficientů soustavy lineárních algebraických rovnic regulární, potom má soustava **právě jedno řešení**.

Funkce pro práci s maticemi

Grafické znázornění výpočtu determinantu matice Sarrusovým pravidlem pro matici rozměru 3x3



Vlastní funkce a operátory

Příklad: Funkce pro řešení soustavy lineárních algebraických rovnic – vstupní parametry: matice koeficientů soustavy a vektor pravých stran.

```
function x = soustava_rovnic(A,b)
x = A \ b;          % operace maticová
end                 % konec funkce
```

Volání funkce pro soustavu:

$3x_1 + 4x_2 = 11$

$2x_1 - 5x_2 = -8$

```
x = soustava_rovnic([3,4;2,-5],[11;-8])
x =
    1.0000
    2.0000
```

Ve funkci **soustava_rovnic**, aby výpočet mohl proběhnout je nutno použít **maticovou** operaci **A \ b**.

Řízení běhu výpočtu

`return` – ukončení funkce

Příklad: Funkce pro řešení soustavy lineárních algebraických rovnic s ošetřením řešitelnosti soustavy – vstupní parametry: matice koeficientů soustavy **`A`** a vektor pravých stran **`b`**.

Bylo uvedeno:

Matice, jejíž determinant je **nenulový**, se nazývá **regulární**.

Je-li matice koeficientů soustavy regulární, tedy **`det(A)≠0`**, potom má soustava **právě jedno řešení**.

Inverzní matici lze vytvořit pouze ke čtvercové matici, která je **regulární** a tedy lze pro výpočet soustavy užít vztahy:
`x = inv(A) * b`, resp. **`x = A \ b`**

Pozn.: *Později si ukážeme techniky, jak dále rozpoznat, zda má soustava nekonečně mnoho řešení či žádné řešení.*

Řízení běhu výpočtu

return – ukončení funkce

Příklad: Funkce pro řešení soustavy lineárních algebraických rovnic s ošetřením řešitelnosti soustavy – vstupní parametry: matice koeficientů soustavy **A** a vektor pravých stran **b**.

```
function soust_rov_det(A,b)
if(det(A)==0)
 disp('Determinant se rovna nule => konec!')
 return     % konec běhu funkce
else
    x = A \ b;
    disp('Reseni soustavy')
    disp(x)
end
end
```

Řízení běhu výpočtu

Pokračování příkladu:

Volání funkce pro soustavu:

$3x_1 + 4x_2 = 11$

$2x_1 - 5x_2 = -8$

```
soust_rov_det([3,4;2,-5],[11;-8])
Reseni soustavy
      1
      2
```

$x_1 + x_2 = 11$

$2x_1 + 2x_2 = -8$

```
soust_rov_det([1,1;2,2],[11;-8])
Determinant se rovna nule => konec!
```

Pozn.:

```
det([3,4;2,-5])
ans =
    -23
```

Pozn.:

```
det([1,1;2,2])
ans =
     0
```

Vlastní funkce a operátory

Příklad: Funkce pro výpočet obsahu kruhu

```
function S = obsah(r)
 S = pi*(r.^2);        % operace prvek po prvku
end                    % konec funkce
```

Volání funkce pro více poloměrů, vypočtou se obsahy všech kruhů najednou – vstupní parametr vektor – nutná operace **prvek po prvku** ve funkci

```
polomery = [1,5,10,20]; % vstupni parametr vektor

obsahy = obsah(polomery)
obsahy =
    3.1416      78.5398     314.1593    1256.6371
```

Vlastní funkce a operátory

Příklad: Funkce pro výpočet objemu kvádru

```
function V = objem(a,b,c)
 V = a.*b.*c;   % operace prvek po prvku
end             % konec funkce
```

Volání funkce pro 3 kvádry o stranách:
a_1 = 1 mm, b_1 = 1 mm, c_1 = 1 mm (vlastně krychlička),
a_2 = 2 mm, b_2 = 5 mm, c_2 = 7 mm,
a_3 = 3 mm, b_3 = 6 mm, c_3 = 8 mm.
Aby mohlo proběhnout násobení vektorů prvek po prvku, je nutná operace **prvek po prvku** ve funkci

```
V = objem([1,2,3],[1,5,6],[1,7,8])
V =
     1    70   144
                    % objemy jednotlivých kvádrů
```

Základy algoritmizace

Algoritmus

- schematický postup pro řešení určitého druhu problémů, který je prováděn pomocí konečného množství přesně definovaných kroků

- dnes se tento pojem používá především v informatice a přírodních vědách, jde o daleko širší pojem
  - kuchyňské recepty
  - návody (jak sestavit skleník atd. ;-)
  - postupy (výrobní i jiné)

- slovo **algoritmus** pochází ze jména perského matematika 9. století Abu Jafar Muhammada ibn Mūsā al-Chwārizmího, který ve svých dílech položil základy algebry (arabské číslice, řešení lineárních a kvadratických rovnic)

Detailněji on-line na http://www.algoritmy.net

Základy algoritmizace

Algoritmus

– postup při tvorbě programu, kterým lze prostřednictvím algoritmu řešit určitý problém.

– etapy

- Formulace problému (požadavky, výchozí hodnoty, požadované výsledky, přesnost řešení)
- Analýza úlohy (je úloha řešitelná, má úloha více řešení?)
- Vytvoření algoritmu
- Sestavení programu (zdrojový text v konkrétním programovacím jazyce)
- Odladění programu (syntaktické chyby, logické chyby)

Detailněji on-line na http://www.algoritmy.net

Základy algoritmizace

Algoritmus

– přesný návod k vykonání dané činnosti, vyřešení jisté úlohy.
– sekvence jednoduchých kroků
– postup, který je:

- konečný (finitní) – po určitém počtu kroků skončí (počet kroků může být libovolně velký (podle rozsahu a hodnot vstupních údajů).
- obecný (hromadný, univerzální) – nemá řešit pouze jediný případ, ale skupinu obdobných problémů.
- jednoznačný (deterministický, podmíněný) – v každém kroku je zcela jasně řečeno, co bude následovat.
- opakovatelný – protože je v každém kroku udáno, co bude následovat, je možné opakovat postup a výsledek bude vždy stejný.
- elementární – skládá se z konečného počtu jednoduchých (elementárních) kroků
- resultativní – má alespoň jeden výstup, veličinu, která je v požadovaném vztahu k zadaným vstupům, a tím tvoří odpověď na problém, který algoritmus řeší
- správný (korektní) – pro všechna přístupná data vede postup ke správnému cíli.

Základy algoritmizace

Vyjádřením algoritmu, programování

Augusta Ada Kingová, hraběnka z Lovelace
(Augusta Ada Byronová)
10. prosince 1815 Londýn – 27. listopadu 1852 tamtéž

- anglická matematička a vynálezkyně strojového programování
- známá především svým detailním popisem fungování Babbageova mechanického počítače (analytického stroje), jehož vývoj podporovala i finančně
- mezi jejími poznámkami k analytickému stroji byl i algoritmus, který je považován za první algoritmus zpracovatelný počítačem, je často uváděna jako první programátorka

Viz např.: https://cs.wikipedia.org/wiki/Ada_Lovelace

Základy algoritmizace

Vyjádřením algoritmu
– slovní popis
– vývojový diagram – grafické znázornění

Např.:
Algoritmus pro výpočet obsahu kruhu – slovní popis

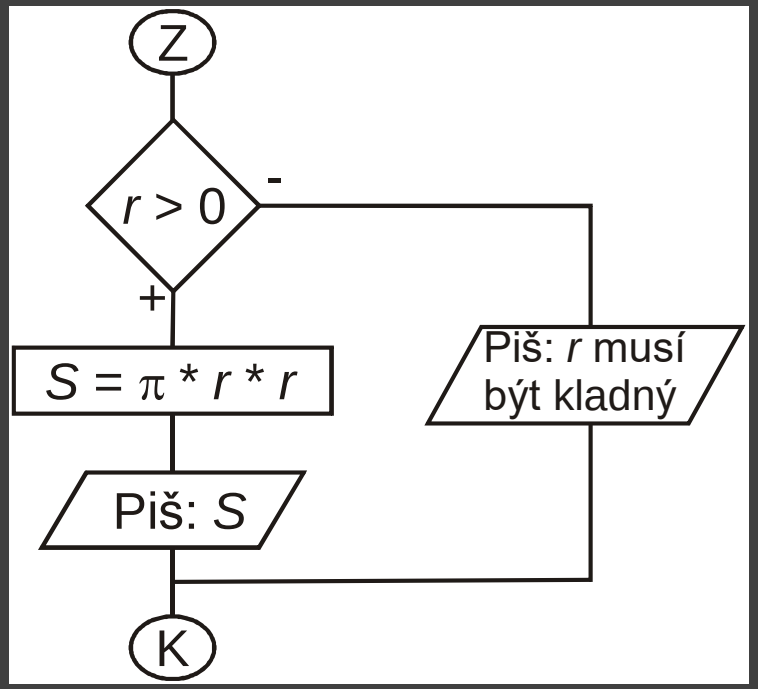
- Formulace problému: vstupní údaj: poloměr r,
  výstupní údaj: obsah kruhu S
- Analýza: $S = \pi r^2$
- Sestavení algoritmu:
  Slovní popis:
  Je poloměr $r > 0$?
  ANO
  `S = pi*r*r`
  Vypiš:
  obsah *S*
  NE
  Vypiš:
  Poloměr r musí být kladný
- Konec

Základy algoritmizace

Vyjádřením algoritmu

Algoritmus pro výpočet obsahu kruhu

Vývojový diagram



Program

```
if (r>0)
   S=pi*r*r;
   disp(S);
else
   disp('r musi byt kladne')
end
```

Viz také: Animace 5 – http://home.zcu.cz/~pkropik/PPEL/slozky/!_zaklady_matlabu

Řízení běhu výpočtu

Řídící příkazy:

- `if` – podmíněný příkaz
- `switch` – přepínač

- `for` – cyklus s daným počtem opakování
- `while` – cyklus s podmínkou na začátku (cyklus bez udání počtu opakování)

Řízení běhu výpočtu

Řídící příkazy:

- `if` – podmíněný příkaz
- `switch` – přepínač

- `for` – cyklus s daným počtem opakování
- `while` – cyklus s podmínkou na začátku (cyklus bez udání počtu opakování)

Viz také: Animace 5 – http://home.zcu.cz/~pkropik/PPEL/slozky/!_zaklady_matlabu

Podmíněný příkaz

```
if logický_výraz
     příkaz;
     příkaz;
     příkaz;
end
```

Podmínka (`logický_výraz`) je výraz, jehož výsledkem je pravda 1 nebo nepravda 0, `příkaz` je proveden, je-li `logický_výraz` pravdivý.

nebo plný tvar:

```
if logický_výraz
 příkazy_je_li_podmínka_pravdivá;
else
 příkazy_není_li_podmínka_pravdivá;
end
```

Středníky nejsou povinné.

Podmíněný příkaz

nebo zcela obecně:

```
if výraz_1
      příkazy_1_1;
      příkazy_1_2;
elseif výraz_2
      příkazy_2_1;
      příkazy_2_2;
      příkazy_2_3;
elseif výraz_3
      příkazy_3_1;
else
      příkazy_4_1;
      příkazy_4_2;
end
```

Příkazy `1_1` a `1_2` proběhnou, je-li pravdivý `výraz_1` .

Příkazy `2_1` až `2_3` proběhnou, je-li pravdivý `výraz_2` .

Příkaz `3_1` proběhne, je-li pravdivý `výraz_3` .

V jiném případě se uskuteční příkazy `4_1` a `4_2`.

Pozor: `else` nebo `elseif` se vztahuje **vždy** k **nejbližšímu** `if` nad ním (samozřejmě pokud se před ním nevyskytne `end`).

Podmíněný příkaz

Příklad: Funkce pro porovnaní čísla s určitou mezí – vstupní parametry: *číslo* a *mez testu*, se kterou se bude číslo porovnávat

```
function porovnani(cislo, mezTestu)
if (cislo < mezTestu)          % relační operátor „je menší“
     disp('cislo je mensi nez zadana mez')
elseif (cislo == mezTestu)  % rovná se ve smyslu porovnání
     disp('cislo je rovno zadane mezi')
else
     disp('cislo je vetsi nez zadana mez')
end                    % konec podmíněného příkazu if
end                    % konec funkce
```

Volání funkce:

```
porovnani(3, 2)
cislo je vetsi nez zadana mez
```

nebo

```
porovnani(5, 5)
cislo je rovno zadane mezi
```

Přepínač

switch
nahradí **if** v případě, kdy mám více možností a **if** by bylo mnohokrát za sebou (speciální případ podmínky).

```
switch výraz
 case případ_1
     příkazy_1_1;
 case případ_2
     příkazy_2_1 ;
     příkazy_2_2;
 case {případ_3,případ_4,případ_5}
     příkazy_3_1;
     příkazy_3_2;
     příkazy_3_3;
 otherwise
    příkazy_jiné ;
end
```

Příkaz 1_1 proběhne, je-li **výraz** roven **případu_1**.

Příkazy 2_1 a 2_2 proběhnou, je-li **výraz** roven **případu_2**.

Příkazy 3_1 až 3_3 proběhnou, je-li **výraz** roven **případu_3**, **případu_4** nebo **případu_5**

Příkazy **jiné** proběhnou, pokud žádný z **případů** neodpovídá **výrazu**.

Přepínač

switch
nahradí **if** v případě, kdy mám více možností a **if** by bylo mnohokrát za sebou (speciální případ podmínky).

```
switch výraz
  case případ_1
      příkazy_1_1;
 case případ_2
      příkazy_2_1 ;
      příkazy_2_2;
 case {případ_3,případ_4,případ_5}
      příkazy_3_1;
      příkazy_3_2;
      příkazy_3_3;
 otherwise
    příkazy_jiné ;
end
```

Počet příkazů ve větvi **case** by měl být malý, pokud potřebujete více příkazů, napište další funkci a tu ve větvi **case** zavolejte

Příkazy **jiné** proběhnou, pokud žádný z **případů** neodpovídá **výrazu**.

Přepínač

Např.: Čísla 1 - 7 značí dny v týdnu - pondělí až neděle

```
volba_dne = 3;
% volba_dne je řídící proměnná příkazu switch
switch volba_dne
      case 1
            příkazy pro pondělí;
      case 3
            příkazy pro středa;
      case 5
            příkazy pro pátek;
      case 6
            příkazy pro sobota;
      otherwise
            příkazy pro všechny jiné varianty;
end;
```

Přepínač

To je totéž jako:

```
volba_dne = 3;                  % přiřazení
if volba_dne == 1               % porovnání na rovnost
     příkazy pro pondělí;
elseif volba_dne == 3           % porovnání na rovnost
     příkazy pro středa;
elseif volba_dne == 5           % porovnání na rovnost
     příkazy pro pátek;
elseif volba_dne == 6           % porovnání na rovnost
     příkazy pro sobota;
else
     příkazy pro všechny jiné varianty;
end;
```

- což není moc přehledné – lepší je zde použít **switch** – **case**.

Přepínač

Podobně: dny v týdnu - pondělí až neděle – textové řetězce musí být v apostrofech ''

```
volba_dne = 'streda';
switch volba_dne
      case 'pondeli'
            příkazy pro pondělí;
      case 'streda'
            příkazy pro středa;
      case 'patek'
            příkazy pro pátek;
      case 'sobota'
            příkazy pro sobota;
      otherwise
            příkazy pro všechny jiné varianty;
end
```

Přepínač

nebo, chci-li ošetřit dva dny najednou:

```
volba_dne = 'streda';
switch volba_dne
      case 'pondeli'
            příkazy pro pondělí;
      case 'streda'
            příkazy pro středa;
      case 'patek'
            příkazy pro pátek;
      case {'sobota', 'nedele'}
            příkazy pro sobota a neděle;
      otherwise
            příkazy pro všechny jiné varianty;
end
```

seznam variant ve složených závorkách

$\{v_1, v_2, v_3, \ldots, v_n\}$

Přepínač

Příklad: Funkce – informace o pracovní činnosti v týdnu

```
function tyden(volba_dne)
 switch volba_dne
   case 'pondeli'
      disp('oddych po víkendu');
   case 'utery'
      disp('příprava na pracovní den');
   case 'streda'
      disp('pracovní den');
   case 'ctvrtek'
      disp('oddych po pracovním dni');
   case 'patek'
      disp('příprava na víkend');
   case {'sobota', 'nedele'}
      disp('víkendové volno');
   otherwise
      disp('tento den neexistuje');
 end   % konec switch
end    % konec funkce
```

vstup musí být řetezec znaků

Volání funkce:

```
tyden('streda')
pracovní den

nebo
tyden('nedele')
víkendové volno

nebo
tyden('rijen')
tento den neexistuje
```

Přepínač

Příklad: Funkce – jednoduchá kalkulačka

```
function kalkulacka(A,B,znak)
% funkce ma 3 vstupy: A,B - ciselné hodnoty, znak – retezec
 switch (znak)
  case '+'
      disp(A+B);
  case '-'
      disp(A-B);
  case '*'
    disp(A.*B);
  case '/'
      disp(A./B);
  case '\'
      disp(A.\B);
  otherwise
      disp('takto nepocitam');
 end   % konec switch
end    % konec funkce
```

Volání funkce:

```
kalkulacka(3,5,'+')
     8
```

nebo

```
kalkulacka(8,4,'/')
     2
```

nebo

```
kalkulacka(8,4,'\')
    0.5000
```

nebo

```
kalkulacka(12,7,'!')
takto nepocitam
```

Vlastní funkce a operátory

Příklad: Funkce – jednoduchá kalkulačka

```
function kalkulacka(A,B,znak)
 switch (znak)
  case '+'
      disp(A+B);
  case '-'
      disp(A-B);
  case '*'
     disp(A.*B); % operace prvek po prvku
  case '/'
      disp(A./B); % operace prvek po prvku
  case '\'
      disp(A.\B); % operace prvek po prvku
  otherwise
      disp('Toto nepocitam');
 end  % konec switch
end   % konec funkce
```

Volání funkce:

```
kalkulacka([8,4,6],[2,1,3],'*')
      16     4     18
```