MATLAB

Úkoly v Matlabu včetně řešení

Vaňha Jiří

Děčín
2025

Obsah

Část I

1. Měsíc - Základní ovládání a syntaxe Cíl : Psát skrypty a funkce bez přemýšlení syntaxe, umět grafy, práce s daty

Kapitola 1

1 Týden - Matice a vektory

1.1 Pondělí : Operace s maticemi

1.1.1 Úkoly

1. Vytvořte matici A rozměru 3×3 obsahující čísla 1 až 9.
2. Přičtěte k ní jednotkovou matici stejné velikosti.
3. Odečtěte od ní matici náhodných čísel 3×3
4. Vynásobte ji po prvcích maticí, kde všechny hodnoty jsou 5.
5. Proveďte maticové násobení s jednotkovou maticí 3×3.
6. Vypočítejte inverzi této matice (pokud existuje).
7. Ověřte, že inverze je správná $A \cdot A^{-1} = I$
8. Najděte determinant matice.
9. Vypočítejte stopu matice (součet diagonálních prvků).
10. Zvyšte všechny prvky o 10.
11. Vynásobte všechny prvky číslem π.
12. Zaokrouhlete výsledky na 2 desetinná místa.
13. Najděte maximum a minimum v celé matici.
14. Najděte součet všech prvků.
15. Najděte součet prvků po řádcích.
16. Najděte součet prvků po sloupcích.
17. Normalizujte všechny prvky na interval [0 , 1] [0,1].
18. Nahraďte všechny hodnoty menší než průměr matici nulou.
19. Vytvořte transponovanou matici.
20. Určete, zda je matice symetrická.

1.1.2 Řešení (kompaktní MATLAB kód)

```
% 1. Vytvoření matice A 3x3 s hodnotami 1..9
A = reshape(1:9, 3, 3);

% 2. Přičtení jednotkové matice
A_plus_I = A + eye(3);

% 3. Odečtení matice náhodných čísel 3x3
R = rand(3);
A_minus_R = A - R;

% 4. Po-prvkové násobení maticí všech pětek
F = 5 * ones(3);
A_elem_mult = A .* F;

% 5. Maticové násobení s jednotkovou maticí
A_mat_mult = A * eye(3);

% 6. Inverze matice (pokud je regulární)
if det(A) ~= 0
A_inv = inv(A);
else
A_inv = NaN;
end

% 7. Ověření inverze
if ~isnan(A_inv)
check_identity = round(A * A_inv, 10); % zaokrouhlení pro
    numerickou stabilitu
else
check_identity = [];
end

% 8. Determinant
A_det = det(A);

% 9. Stop (trace)
A_trace = trace(A);

% 10. Zvýšení všech prvků o 10
A_plus_10 = A + 10;

% 11. Násobení prvků pi
A_times_pi = A * pi;

% 12. Zaokrouhlení na 2 desetinná místa
A_rounded = round(A_times_pi, 2);

% 13. Maximum a minimum
A_max = max(A(:));
```

```
A_min = min(A(:));

% 14. Součet všech prvků
A_sum_all = sum(A(:));

% 15. Součet po řádcích
A_sum_rows = sum(A, 2); % sloupcový vektor sčítání po řádcích

% 16. Součet po sloupcích
A_sum_cols = sum(A, 1); % řádkový vektor sčítání po sloupcích

% 17. Normalizace na [0,1]
A_minval = A_min;
A_maxval = A_max;
A_norm = (A - A_minval) / (A_maxval - A_minval);

% 18. Nahrazení hodnot menších než průměr nulou
A_mean = mean(A(:));
A_thresh = A;
A_thresh(A_thresh < A_mean) = 0;

% 19. Transponovaná matice
A_T = A.';

% 20. Kontrola symetrie
is_symmetric = isequal(A, A_T);

% (Volitelně: výpis výsledků)
disp('A ='), disp(A)
disp('A_plus_I ='), disp(A_plus_I)
disp('A_minus_R ='), disp(A_minus_R)
disp('A_elem_mult ='), disp(A_elem_mult)
disp('A_inv ='), disp(A_inv)
disp('check_identity ='), disp(check_identity)
disp(['det(A) = ', num2str(A_det)])
disp(['trace(A) = ', num2str(A_trace)])
disp('A_rounded ='), disp(A_rounded)
disp(['max = ', num2str(A_max), ', min = ', num2str(A_min)])
disp(['sum all = ', num2str(A_sum_all)])
disp('sum rows ='), disp(A_sum_rows)
disp('sum cols ='), disp(A_sum_cols)
disp('A_norm ='), disp(A_norm)
disp(['mean = ', num2str(A_mean)])
disp('A_thresh ='), disp(A_thresh)
disp(['is_symmetric = ', mat2str(is_symmetric)])
```

1.1.3 Řešení s podrobným vysvětlením kódu

1. **Vytvoření matice A**:

```
A = reshape(1:9, 3, 3);
```

Vysvětlení: `1:9` vytvoří vektor [1 , 2 , . . . , 9] [1,2,. . . ,9]. `reshape(...,3,3)` tento vektor přetvaruje do matice 3 × 3 3×3 plně po sloupcích (Matlab naplňuje sloupce nejprve). Pokud chcete naplňovat po řádcích, použijte `reshape(1:9, 3, 3).'` nebo jinou permutaci.

2. **Přičtení jednotkové matice**:

```
A_plus_I = A + eye(3);
```

Vysvětlení: `eye(3)` vytvoří identitu 3 I 3

. Operace `A + eye(3)` provede element-wise součet, protože matice jsou stejného rozměru.

3. **Odečtení náhodné matice**:

```
R = rand(3);
A_minus_R = A - R;
```

Vysvětlení: `rand(3)` generuje 3 × 3 3×3 matice s hodnotami v rozmezí [0 , 1) [0,1). Odečtení probíhá element-wise.

4. **Prvkové násobení maticí všech pětek**:

```
F = 5 * ones(3);
A_elem_mult = A .* F;
```

Vysvětlení: `ones(3)` vytvoří matici plnou jedniček; vynásobením 5 získáme matici pětek. Operátor . provádí násobení *po prvcích*. (Pokud byste použili , Matlab by interpretoval jako maticové násobení — dimenze by musely sedět.)

5. **Maticové násobení s identitou**:

```
A_mat_mult = A * eye(3);
```

Vysvětlení: Násobení maticí identity by mělo vrátit původní matici (A * I = A). Toto demonstruje rozdíl mezi a ..

6. **Inverze matice**:

```
if det(A) ~= 0
A_inv = inv(A);
else
A_inv = NaN;
end
```

Vysvětlení: Matice je invertovatelná právě tehdy, když její determinant není nulový. Používáme `det(A)` pro kontrolu. Funkce `inv(A)` vrací inverzní matici. Pozor: pro numericky robustnější řešení systému = Ax=b se doporučuje nepoužívat explicitně `inv` (lepší je řešit pomocí `x = A b`).

7. **Ověření inverze**:

```
check_identity = round(A * A_inv, 10);
```

Vysvětlení: V reálných výpočtech se mohou objevit drobné numerické chyby (např. 1.0×10 16 1.0×10 16). Proto `round(...,10)` zaokrouhlí prvky pro porovnání.

8. **Determinant**:

```
A_det = det(A);
```

Vysvětlení: Determinant dává informaci o tom, zda je matice singularní (det = 0) a o škálování objemu lineární transformace.

9. **Stopa (trace)**:

```
A_trace = trace(A);
```

Vysvětlení: `trace` vrací součet diagonálních prvků; užitečné ve výpočtech lineární algebry a statistikách (např. stopa matice kovariance).

10. **Zvýšení všech prvků o 10**:

```
A_plus_10 = A + 10;
```

Vysvětlení: Skalární se přičte ke všem prvkům (implicitní rozšíření).

11. **Násobení** :

```
A_times_pi = A * pi;
```

Vysvětlení: Matice se násobí skalárem.

12. **Zaokrouhlení na 2 desetinná místa**:

```
A_rounded = round(A_times_pi, 2);
```

Vysvětlení: Funkce `round(X,n)` zaokrouhlí všechny prvky matice na n desetinných míst.

13. **Maximum a minimum**:

```
A_max = max(A(:));
A_min = min(A(:));
```

Vysvětlení: `A(:)` převádí matici na sloupcový vektor; následné `max`/`min` najdou extrémy přes všechny prvky.

14. **Součet všech prvků**:

```
A_sum_all = sum(A(:));
```

Vysvětlení: Suma přes všechny prvky; alternativně lze použít `sum(sum(A))`.

15. **Součet po řádcích a sloupcích**:

```
A_sum_rows = sum(A, 2);
A_sum_cols = sum(A, 1);
```

Vysvětlení: Druhý parametr udává dimenzi, přes kterou se sumuje — `2` znamená součet přes sloupce (výsledek: sloupcový vektor), `1` znamená součet přes řádky.

16. **Normalizace na interval [0 , 1] [0,1]**:

```
A_norm = (A - A_minval) / (A_maxval - A_minval);
```

Vysvětlení: Standardní min-max normalizace: všechna data posuneme o minimum a dělíme rozptylem (max-min). Po této operaci jsou všechny hodnoty v [0 , 1] [0,1].

17. **Nahrazení hodnot menších než průměr nulou**:

```
A_mean = mean(A(:));
A_thresh(A_thresh < A_mean) = 0;
```

Vysvětlení: Tvoříme masku prvků menších než průměr a přiřazujeme 0 — typický způsob prahování v MATLABu.

18. **Transponovaná matice**:

```
A_T = A.';
```

Vysvětlení: Operátor `.'` provádí nekomplexní transpozici (pokud pracujete s komplexními čísly a chcete konjugovanou transpozici, použijte `A'`).

19. **Kontrola symetrie**:

```
is_symmetric = isequal(A, A_T);
```

Vysvětlení: `isequal` porovnává přesnou shodu prvků. Pro numerické matice s plovoucí čárkou bude lepší kontrola s tolerancí, např.

```
is_symmetric_tol = max(abs(A - A_T), [], 'all') < 1e-10;
```

což zohlední malé numerické odchylky.

Doplňkové poznámky a tipy pro cvičení:

- Pro testování různých kroků (např. jiného naplnění matice) zkuste `reshape(1:9,3,3).'` aby se data naplnila po řádcích.
- Místo `inv(A)` preferujte `A\b` při řešení rovnic = Ax=b — je to numericky stabilnější a rychlejší.

- Při porovnávání matic používejte buď `isequal` pro přesné srovnání, nebo tolerantní porovnání s `norm(A-B)` nebo `max(abs(A-B),[],'all')`.
- Pokud byste potřeboval (pro dokumentaci nebo výukové účely) vypisovat matici do LaTeXu, použijte `mat2str(A)` nebo funkce pro export (CSV).

1.2 Úterý : Indexování matic

1.2.1 Úkoly

1. Vytvořte matici B rozměru 4×5 $4{\times}5$ s náhodnými celými čísly od 1 do 20.
2. Získejte prvek v řádku 2 a sloupci 3.
3. Získejte celý 3. řádek.
4. Získejte celý 4. sloupec.
5. Použijte lineární indexování a vyberte 7. prvek matice.
6. Vyberte všechny sudé prvky matice.
7. Vyberte všechny prvky větší než 10.
8. Změňte všechny prvky menší než 5 na 0.
9. Najděte indexy všech prvků větších než 15.
10. Vyberte první dva řádky a poslední tři sloupce.
11. Vyberte diagonální prvky matice (pokud má matice nečtvercový tvar, berte hlavní diagonálu).
12. Vytvořte masku pro prvky mezi 5 a 15 a vyberte je.
13. Inverzní maska: vyberte všechny prvky, které nejsou mezi 5 a 15.
14. Nahraďte všechny prvky větší než 12 průměrem matice.
15. Proveďte permutaci řádků (vyměňte první a poslední řádek).
16. Proveďte permutaci sloupců (vyměňte druhý a čtvrtý sloupec).
17. Získejte podmatici složenou z lichých řádků a sudých sloupců.
18. Použijte logické indexování k nastavení všech prvků menších než 8 na -1.
19. Změňte hodnotu prostředního prvku (pro lichý počet řádků a sloupců) na 999.
20. Zkontrolujte, zda existuje prvek rovný 20.

1.2.2 Řešení (kompaktní MATLAB kód)

```
% 1. Náhodná matice B 4x5 s celými čísly 1..20
B = randi(20, 4, 5);

% 2. Prvek v řádku 2, sloupec 3
elem_2_3 = B(2,3);

% 3. Celý 3. řádek
row3 = B(3,:);

% 4. Celý 4. sloupec
col4 = B(:,4);

% 5. Lineární indexování (7. prvek)
lin7 = B(7);

% 6. Sudé prvky
even_elems = B(mod(B,2)==0);

% 7. Prvky větší než 10
gt10 = B(B>10);

% 8. Menší než 5 -> 0
B(B<5) = 0;

% 9. Indexy prvků větších než 15
idx_gt15 = find(B>15);

% 10. Podmatice první 2 řádky, poslední 3 sloupce
sub_B = B(1:2,end-2:end);

% 11. Diagonální prvky (hlavní diagonála)
diag_B = diag(B);

% 12. Maska pro prvky mezi 5 a 15
mask_5_15 = B>=5 & B<=15;
B_masked = B(mask_5_15);

% 13. Inverzní maska
B_masked_inv = B(~mask_5_15);

% 14. Nahrazení prvků větších než 12 průměrem
B(B>12) = mean(B(:));

% 15. Permutace řádků (1.      poslední)
B([1,end],:) = B([end,1],:);

% 16. Permutace sloupců (2.      4.)
B(:,[2,4]) = B(:,[4,2]);

```

```
% 17. Podmatice liché řádky, sudé sloupce
sub_B2 = B(1:2:end,2:2:end);

% 18. Logické indexování: prvky < 8 -> -1
B(B<8) = -1;

% 19. Prostřední prvek -> 999
mid_row = ceil(size(B,1)/2);
mid_col = ceil(size(B,2)/2);
B(mid_row, mid_col) = 999;

% 20. Kontrola, zda existuje prvek rovný 20
exist20 = any(B(:) == 20);

disp('B ='), disp(B)
disp('elem_2_3 ='), disp(elem_2_3)
disp('row3 ='), disp(row3)
disp('col4 ='), disp(col4)
disp('lin7 ='), disp(lin7)
disp('even_elems ='), disp(even_elems)
disp('gt10 ='), disp(gt10)
disp('idx_gt15 ='), disp(idx_gt15)
disp('sub_B ='), disp(sub_B)
disp('diag_B ='), disp(diag_B)
disp('B_masked ='), disp(B_masked)
disp('B_masked_inv ='), disp(B_masked_inv)
disp('exist20 ='), disp(exist20)
```

Listing 1.1: Úterý řešení — kompletní skript

1.2.3 Řešení s podrobným vysvětlením kódu

1. **Vytvoření matice**:

```
B = randi(20, 4, 5);
```

Vysvětlení: `randi(20,4,5)` vytvoří 4×5 matici s náhodnými celými čísly od 1 do 20.

2. **Výběr prvku v řádku 2 a sloupci 3**:

```
elem_2_3 = B(2,3);
```

Vysvětlení: Standardní indexování matic, první parametr je řádek, druhý sloupec.

3. **Výběr řádku a sloupce**:

```
row3 = B(3,:);
col4 = B(:,4);
```

Vysvětlení: Dvojtečka (`:`) vybere všechny prvky v dané dimenzi.

4. **Lineární indexování**:

```
lin7 = B(7);
```

Vysvětlení: MATLAB ukládá matici po sloupcích, lineární index 7 odpovídá 7. prvku při sloupcovém čtení.

5. **Logické a podmínkové indexování**:

```
even_elems = B(mod(B,2)==0);
gt10 = B(B>10);
B(B<5) = 0;
```

Vysvětlení: Logický vektor (masky) vybírá nebo modifikuje prvky splňující podmínku.

6. **Indexování pomocí `find`**:

```
idx_gt15 = find(B>15);
```

Vysvětlení: Vrací lineární indexy všech prvků větších než 15.

7. **Výběr podmatic a diagonál**:

```
sub_B = B(1:2,end-2:end);
diag_B = diag(B);
```

Vysvětlení: Výběr řádků/sloupců podle rozsahu; `diag` vrací hlavní diagonálu.

8. **Masky a inverzní masky**:

```
mask_5_15 = B>=5 & B<=15;
B_masked = B(mask_5_15);
B_masked_inv = B(~mask_5_15);
```

Vysvětlení: Kombinace logických podmínek; invertuje logickou matici.

9. **Nahrazení a permutace prvků**:

```
B(B>12) = mean(B(:));
B([1,end],:) = B([end,1],:);
B(:,[2,4]) = B(:,[4,2]);
```

Vysvětlení: Skalární nahrazení; výměna řádků a sloupců jednoduchou indexací.

10. **Výběr podmatic lichých/sudých řádků/sloupců**:

```
sub_B2 = B(1:2:end,2:2:end);
```

Vysvětlení: 1:2:end vybírá každý druhý řádek/sloupec.

11. **Logické nastavení prvků a úprava prostředního prvku**:

```
B(B<8) = -1;
mid_row = ceil(size(B,1)/2);
mid_col = ceil(size(B,2)/2);
B(mid_row, mid_col) = 999;
```

Vysvětlení: Indexování podle masky; `ceil(size(B)/2)` najde prostřední index.

12. **Kontrola existence prvku**:

```
exist20 = any(B(:) == 20);
```

Vysvětlení: `any` vrací true, pokud je podmínka splněna alespoň pro jeden prvek.

Tipy:

- Lineární a logické indexování jsou základem pro složitější manipulace s maticemi.
- Masky a podmatice lze kombinovat pro selektivní úpravy dat.
- Pro dynamické výběry a úpravy matic se doporučuje kombinovat `find`, logické masky a rozsahy.

1.3 Středa : Tvorba speciálních matic

1.3.1 Úkoly

1. Vytvořte nulovou matici 3×4.
2. Vytvořte jednotkovou matici 4×4.
3. Vytvořte diagonální matici s hodnotami 1,2,3,4 na diagonále.
4. Vytvořte magickou matici 3×3.
5. Vytvořte náhodnou matici 5×5 s hodnotami mezi 0 a 1.
6. Vytvořte náhodnou celočíselnou matici 4×4 od 10 do 50.
7. Vytvořte matici velikosti 3×3 se všemi hodnotami 7.
8. Vytvořte matici s postupně rostoucími hodnotami 1–12 po řádcích.
9. Vytvořte matici s postupně rostoucími hodnotami 1–12 po sloupcích.
10. Vytvořte 3×3 matici identických náhodných hodnot (např. všechny prvky stejné náhodné číslo).
11. Vytvořte horní trojúhelníkovou matici z libovolné 4×4 matice.
12. Vytvořte dolní trojúhelníkovou matici z libovolné 4×4 matice.
13. Vytvořte 4×4 matici s hodnotami 0,1,2,3 po diagonále (funkce `diag`).
14. Vytvořte opakující se blokovou matici [1 2;3 4] 3×3.
15. Vytvořte vektor s hodnotami -5 až 5 a přetvořte ho na 3×4 matici (reshape).
16. Vytvořte 3×3 matici se stejnými hodnotami v řádcích (např. první řádek 1, druhý 2, třetí 3).
17. Vytvořte 3×3 matici se stejnými hodnotami ve sloupcích (např. první sloupec 1, druhý 2, třetí 3).
18. Vytvořte náhodnou binární matici 5×5 (0 a 1).
19. Vytvořte identitu 3×3 a násobte ji náhodnou maticí 3×3.
20. Vytvořte diagonální matici, vypište její diagonálu a spočítejte její inverzi (pokud existuje).

1.3.2 Řešení (kompaktní MATLAB kód)

```
% 1. Nulová matice 3x4
Z = zeros(3,4);

% 2. Jednotková matice 4x4
I = eye(4);

% 3. Diagonální matice s hodnotami 1,2,3,4
D = diag(1:4);

% 4. Magická matice 3x3
M = magic(3);

% 5. Náhodná matice 5x5 [0,1]
R = rand(5);

% 6. Náhodná celočíselná matice 4x4 10..50
RI = randi([10 50],4,4);

% 7. Matice 3x3 všechny hodnoty 7
F = 7*ones(3);

% 8. Postupně rostoucí hodnoty 1-12 po řádcích
A_row = reshape(1:12,4,3)'; % transpose pro řádkové naplnění

% 9. Postupně rostoucí hodnoty 1-12 po sloupcích
A_col = reshape(1:12,3,4);

% 10. Identické náhodné číslo
x = rand;
B_same = x * ones(3);

% 11. Horní trojúhelníková matice
U = triu(RI);

% 12. Dolní trojúhelníková matice
L = tril(RI);

% 13. Diagonála 0,1,2,3
D2 = diag(0:3);

% 14. Opakující se bloková matice 3x3
blk = repmat([1 2;3 4], 2,2); % bude větší než 3x3, lze oříznout

% 15. Vektor -5..5     matice 3x4
v = -5:1:6;
B = reshape(v,3,4);

% 16. Stejné hodnoty v řádcích
row_mat = repmat((1:3)',1,3);
```

```
50
51 % 17. Stejné hodnoty ve sloupcích
52 col_mat = repmat(1:3,3,1);
53
54 % 18. Náhodná binární matice 5x5
55 bin = randi([0 1],5,5);
56
57 % 19. Identita * náhodná matice
58 I3 = eye(3);
59 R3 = rand(3);
60 prod = I3 * R3;
61
62 % 20. Diagonální matice, výpis diagonály, inverze
63 D3 = diag([2 3 4]);
64 diag_vals = diag(D3);
65 if det(D3) ~= 0
66 D3_inv = inv(D3);
67 else
68 D3_inv = NaN;
69 end
70
71 disp('Z ='), disp(Z)
72 disp('I ='), disp(I)
73 disp('D ='), disp(D)
74 disp('M ='), disp(M)
75 disp('R ='), disp(R)
76 disp('RI ='), disp(RI)
77 disp('F ='), disp(F)
78 disp('A_row ='), disp(A_row)
79 disp('A_col ='), disp(A_col)
80 disp('B_same ='), disp(B_same)
81 disp('U ='), disp(U)
82 disp('L ='), disp(L)
83 disp('D2 ='), disp(D2)
84 disp('blk ='), disp(blk)
85 disp('B ='), disp(B)
86 disp('row_mat ='), disp(row_mat)
87 disp('col_mat ='), disp(col_mat)
88 disp('bin ='), disp(bin)
89 disp('prod ='), disp(prod)
90 disp('D3_inv ='), disp(D3_inv)
91 disp('diag_vals ='), disp(diag_vals)
```

Listing 1.2: Středa řešení — kompletní skript

1.3.3 Řešení s podrobným vysvětlením kódu

1. **Nulová a jednotková matice:**

```
1 Z = zeros(3,4);
2 I = eye(4);
```

Vysvětlení: `zeros(m,n)` vytváří m×n nulovou matici, `eye(n)` jednotkovou čtvercovou matici.

2. **Diagonální matice:**

```
D = diag(1:4);
```

Vysvětlení: `diag(v)` vytvoří matici s hodnotami vektor `v` na hlavní diagonále.

3. **Magická matice:**

```
M = magic(3);
```

Vysvětlení: `magic(n)` vytvoří n×n čtvercovou matici, kde součet každého řádku, sloupce a diagonály je stejný.

4. **Náhodné matice:**

```
R = rand(5);
RI = randi([10 50],4,4);
B_same = rand*ones(3);
```

Vysvětlení: `rand` generuje reálná čísla 0–1, `randi([a b],m,n)` celá čísla a `ones*rand` duplikuje náhodné číslo.

5. **Postupně rostoucí hodnoty:**

```
A_row = reshape(1:12,4,3)';
A_col = reshape(1:12,3,4);
```

Vysvětlení: `reshape` přetvoří vektor na matici; transpose (') zajistí řádkové naplnění.

6. **Horní a dolní trojúhelníkové matice:**

```
U = triu(RI);
L = tril(RI);
```

Vysvětlení: `triu` vybere horní trojúhelník (včetně diagonály), `tril` dolní.

7. **Blokové matice a repmat:**

```
blk = repmat([1 2;3 4],2,2);
```

Vysvětlení: `repmat(A,m,n)` opakuje matici A m×n krát.

8. **Diagonála a inverze:**

```
D3 = diag([2 3 4]);
diag_vals = diag(D3);
D3_inv = inv(D3);
```

Vysvětlení: Inverze diagonální matice je jednoduchá: převrácená hodnota na diagonále.

1.4 Čtvrtek : Manipulace s rozměry (reshape, permute, squeeze)

1.4.1 Úkoly

1. Vytvořte vektor 1–12 a přetvořte jej na matici 3x4.
2. Přetvořte stejný vektor na tenzor 2×2×3.
3. Použijte `reshape`, abyste vytvořili matici 6×2.
4. Vytvořte náhodnou 3D matici velikosti 3×4×2 a vypište její rozměry.
5. Přehodťe dimenze 3×4×2 na 4×3×2 pomocí `permute`.
6. Z 3D matice 3×4×2 udělejte 2×12 matici pomocí `reshape`.
7. Vytvořte 4×4 matici a otočte ji pomocí `transpose` a `permute`.
8. Použijte `rot90` k otočení matice o 90°.
9. Vytvořte matici 1×3×1×4 a odstraňte dimenze 1 pomocí `squeeze`.
10. Přidejte do matice 3×3 novou dimenzi pomocí `reshape` (udělejte z ní 3×3×1).
11. Použijte `reshape`, abyste převedli vektor 1:24 na čtyřrozměrnou matici 2×3×2×2.
12. Vytvořte náhodný vektor délky 9 a přetvořte jej na matici 3×3.
13. Vytvořte náhodnou matici 2×3×4 a změňte pořadí dimenzí na 3×4×2.
14. Vytvořte matici 4×5 a udělejte z ní vektor 20 prvků.
15. Použijte `reshape`, abyste matici 5×2 přetvořili na 2×5.
16. Zkombinujte dvě dimenze tenzoru 2×3×4 do tvaru 6×4.
17. Rozdělte vektor 1:16 na bloky 2×2 pomocí `reshape`.
18. Vytvořte čtyřrozměrnou matici a pak si zobrazte jen její velikosti dimenzí pomocí `size`.
19. Použijte `permute`, abyste prohodili první a poslední dimenzi matice 2×3×4.
20. Vytvořte 3D matici 5×1×4 a odstraňte nadbytečnou dimenzi pomocí `squeeze`.

1.4.2 Řešení (kompaktní MATLAB kód)

```
v = 1:12;
A1 = reshape(v,3,4);        % 1
A2 = reshape(v,2,2,3);      % 2
A3 = reshape(v,6,2);        % 3
X = rand(3,4,2);            % 4
sz = size(X);               % velikost
Xp = permute(X,[2 1 3]);    % 5
Xr = reshape(X,2,12);       % 6
B = rand(4);                % 7
B_t = B';                   % transpose
B_p = permute(B,[2 1]);
B_rot = rot90(B);           % 8
Y = reshape(1:12,[1 3 1 4]);
Y2 = squeeze(Y);            % 9
Z = reshape(1:9,[3 3 1]); % 10
W = reshape(1:24,[2 3 2 2]); % 11
r = rand(1,9); M = reshape(r,3,3); % 12
T = rand(2,3,4);
T2 = permute(T,[2 3 1]);  % 13
C = rand(4,5); vC = C(:); % 14
D = reshape(rand(5,2),2,5); % 15
E = rand(2,3,4);
E2 = reshape(E,6,4);      % 16
blk = reshape(1:16,2,2,4); % 17
size_blk = size(blk);
Q = permute(E,[3 2 1]);   % 18
F = rand(5,1,4);
F2 = squeeze(F);            % 19
```

Listing 1.3: Čtvrtek řešení — kompletní skript

1.4.3 Řešení s podrobným vysvětlením kódu

1. **Z vektoru na matici:**

```
v = 1:12;
A1 = reshape(v,3,4);
```

Vysvětlení: `reshape(v,m,n)` přetvoří vektor do tvaru m×n. Zde z 1–12 dostaneme 3 řádky a 4 sloupce.

2. **Vektor na tenzor:**

```
A2 = reshape(v,2,2,3);
```

Vysvětlení: Rozměry se musí násobit do celkového počtu prvků ($2\times2\times3 = 12$).

3. **Další varianta reshape:**

```
A3 = reshape(v,6,2);
```

Vysvětlení: Získáme 6 řádků, 2 sloupce.

4. **Rozměry 3D matice:**

```
X = rand(3,4,2);
sz = size(X);
```

Vysvětlení: Funkce `size` vrátí vektor rozměrů, zde [3 4 2].

5. **Prohození dimenzí:**

```
Xp = permute(X,[2 1 3]);
```

Vysvětlení: `permute` mění pořadí dimenzí – první a druhou jsme prohodili.

6. **Z 3D na 2D:**

```
Xr = reshape(X,2,12);
```

Vysvětlení: $3\times4\times2 = 24$ prvků, můžeme je uspořádat jako 2×12.

7. **Transpose vs permute:**

```
B_t = B';
B_p = permute(B,[2 1]);
```

Vysvětlení: `B'` transponuje, `permute` je obecnější (užitečné pro více dimenzí).

8. **Otočení matice:**

```
B_rot = rot90(B);
```

Vysvětlení: Funkce `rot90` rotuje matici o 90°.

9. **Squeeze:**

```
Y = reshape(1:12,[1 3 1 4]);
Y2 = squeeze(Y);
```

Vysvětlení: `squeeze` odstraní dimenze délky 1, z (1×3×1×4) se stane (3×4).

10. **Přidání dimenze:**

```
Z = reshape(1:9,[3 3 1]);
```

Vysvětlení: Přidáním 3. dimenze dostaneme 3×3×1, což se často hodí pro 3D operace.

11. **4D matice:**

```
W = reshape(1:24,[2 3 2 2]);
```

Vysvětlení: Vektor 1–24 se přeuspořádá do čtyřrozměrného tvaru.

12. **Náhodná matice 3×3:**

```
r = rand(1,9); M = reshape(r,3,3);
```

13. **Prohazování dimenzí:**

```
T = rand(2,3,4);
T2 = permute(T,[2 3 1]);
```

Vysvětlení: Rozměry (2,3,4) se změní na (3,4,2).

14. **Vektor z matice:**

```
vC = C(:);
```

Vysvětlení: Operátor (:) převádí libovolnou matici do sloupcového vektoru.

15. **5×2 → 2×5:**

```
D = reshape(rand(5,2),2,5);
```

16. **Kombinace dimenzí:**

```
E2 = reshape(E,6,4);
```

Vysvětlení: Z (2×3×4) spojíme první dvě dimenze → (6×4).

17. **Rozdělení do bloků:**

```
blk = reshape(1:16,2,2,4);
```

Vysvětlení: Čtyři bloky 2×2 z vektoru 1–16.

18. **Size na 4D:**

```
size_blk = size(blk);
```

19. **Prohození první a poslední dimenze:**

```
Q = permute(E,[3 2 1]);
```

20. **Odstranění dimenze 1:**

```
F = rand(5,1,4);
F2 = squeeze(F);
```

Vysvětlení: Výsledná velikost je 5×4.

1.5 Pátek : Operace s maticemi (sčítání, násobení, inverze, determinanty)

1.5.1 Úkoly

1. Sečtěte dvě náhodné matice 3×3.
2. Odečtěte matici B od matice A (obě 3×3).
3. Proveďte násobení dvou matic 3×3.
4. Vynásobte matici 3×2 a 2×4.
5. Násobte po prvcích dvě matice 3×3.
6. Vynásobte matici číslem (skalárem).
7. Vypočtěte determinant matice 3×3.
8. Vypočtěte determinant matice 4×4.
9. Najděte inverzi matice 3×3.
10. Ověřte, že $A \cdot A^{-1} = I$.
11. Spočítejte stopu (trace) matice 4×4.
12. Vypočtěte vlastní čísla matice 3×3.
13. Vypočtěte vlastní vektory matice 3×3.
14. Vypočtěte hodnost (rank) matice.
15. Vypočtěte Frobeniovu normu matice.
16. Spočítejte $A^T A$ pro náhodnou matici 3×2.
17. Spočítejte LU rozklad čtvercové matice.
18. Spočítejte QR rozklad matice 3×3.
19. Spočítejte SVD rozklad matice 3×2.
20. Řešte soustavu $Ax = b$ pomocí operátoru zpětného lomítka.

1.5.2 Řešení (kompaktní MATLAB kód)

```
A = rand(3); B = rand(3);
C1 = A + B;                % 1
C2 = A - B;                % 2
C3 = A * B;                % 3
M1 = rand(3,2); M2 = rand(2,4);
C4 = M1*M2;                % 4
C5 = A .* B;               % 5
C6 = 5*A;                  % 6
detA = det(A);             % 7
D = rand(4); detD = det(D); % 8
Ainv = inv(A);             % 9
I_test = A*Ainv;           % 10
trD = trace(D);            % 11
eigvals = eig(A);          % 12
[V,lambda] = eig(A);       % 13
rA = rank(A);              % 14
normA = norm(A,'fro');     % 15
M3 = rand(3,2); C7 = M3'*M3; % 16
[L,U] = lu(A);             % 17
[Q,R] = qr(A);             % 18
[U_s,S,V_s] = svd(M3);     % 19
b = rand(3,1); x = A\b;    % 20
```

Listing 1.4: Pátek řešení — kompletní skript

1.5.3 Řešení s podrobným vysvětlením kódu

1. **Sčítání matic:**

```
A = rand(3); B = rand(3);
C1 = A + B;
```

Vysvětlení: Pro element-wise součet musí mít matice stejné rozměry.

2. **Rozdíl matic:**

```
C2 = A - B;
```

3. **Matice × matice:**

```
C3 = A * B;
```

Vysvětlení: Standardní maticové násobení.

4. **Nečtvercové matice:**

```
M1 = rand(3,2); M2 = rand(2,4);
C4 = M1*M2;
```

Vysvětlení: Podmínka – vnitřní rozměry musí sedět ($2 = 2$).

5. **Násobení po prvcích:**

```
C5 = A .* B;
```

Vysvětlení: Operátor `.*` násobí prvek po prvku.

6. **Skalární násobení:**

```
C6 = 5*A;
```

7. **Determinant 3×3:**

```
detA = det(A);
```

8. **Determinant 4×4:**

```
D = rand(4); detD = det(D);
```

9. **Inverze:**

```
Ainv = inv(A);
```

10. **Kontrola inverze:**

```
I_test = A*Ainv;
```

Vysvětlení: Výsledek by měl být jednotková matice.

11. **Stopa:**

```
trD = trace(D);
```

Vysvětlení: Součet diagonálních prvků.

12. **Vlastní čísla:**

```
eigvals = eig(A);
```

13. **Vlastní čísla + vektory:**

```
[V,lambda] = eig(A);
```

14. **Hodnost:**

```
rA = rank(A);
```

15. **Norma:**

```
normA = norm(A,'fro');
```

Vysvětlení: Frobeniova norma = odmocnina z (součet čtverců všech prvků).

16. **Výraz $A^T A$:**

```
M3 = rand(3,2);
C7 = M3'*M3;
```

17. **LU rozklad:**

```
[L,U] = lu(A);
```

18. **QR rozklad:**

```
[Q,R] = qr(A);
```

19. **SVD rozklad:**

```
[U_s,S,V_s] = svd(M3);
```

20. **Řešení soustavy:**

```
b = rand(3,1);
x = A\b;
```

Vysvětlení: Operátor \ je numericky stabilní způsob, jak řešit $Ax = b$.

1.6 Sobota : Projekt – Generování a vizualizace 2D funkce $z = \sin(x)\cos(y)$

1.6.1 Úkoly

1. Vytvořte vektor x od -5 do 5 s krokem 0.1.
2. Vytvořte vektor y od -5 do 5 s krokem 0.1.
3. Použijte `meshgrid` pro vytvoření 2D sítě souřadnic (X, Y).
4. Definujte funkci $Z = \sin(X). * cos(Y)$.
5. Zobrazte Z pomocí `surf`.
6. Použijte `mesh` místo `surf`.
7. Přidejte barevnou mapu pomocí `colormap(jet)`.
8. Přidejte barevný pruh (`colorbar`).
9. Změňte pohled kamery pomocí `view(45,30)`.
10. Vykreslete konturový graf pomocí `contour`.
11. Vykreslete vyplněný konturový graf pomocí `contourf`.
12. Přidejte popisky os a nadpis.
13. Přidejte mřížku (`grid on`).
14. Upravte tloušťku čar grafu (`LineWidth`).
15. Upravte hustotu sítě (např. krok 0.05 místo 0.1).
16. Změňte rozsah os na $[-\pi, \pi]$.
17. Zobrazte současně 3D povrch a jeho konturu v rovině XY (`surfc`).
18. Použijte průhlednost povrchu (`alpha`).
19. Exportujte obrázek do souboru PNG.
20. Vytvořte funkci, která vykreslí $z = \sin(x)\cos(y)$ pro libovolný rozsah a hustotu.

1.6.2 Řešení (kompaktní MATLAB kód)

```
x = -5:0.1:5; y = -5:0.1:5;                % 1,2
[X,Y] = meshgrid(x,y);                      % 3
Z = sin(X).*cos(Y);                         % 4
figure; surf(X,Y,Z);                        % 5
figure; mesh(X,Y,Z);                        % 6
colormap(jet);                              % 7
colorbar;                                   % 8
view(45,30);                                % 9
figure; contour(X,Y,Z);                     % 10
figure; contourf(X,Y,Z);                    % 11
xlabel('x'); ylabel('y'); zlabel('z');      % 12
title('z = sin(x)cos(y)');                  % 12
grid on;                                    % 13
surf(X,Y,Z,'LineWidth',2);                  % 14
x2 = -5:0.05:5; y2 = -5:0.05:5;             % 15
[X2,Y2] = meshgrid(x2,y2);
Z2 = sin(X2).*cos(Y2);
surf(X2,Y2,Z2);
xlim([-pi pi]); ylim([-pi pi]);             % 16
figure; surfc(X,Y,Z);                       % 17
alpha(0.7);                                 % 18
saveas(gcf,'surface.png');                  % 19
% 20: funkce
function plotSineCos(xrange,yrange,step)
  [X,Y] = meshgrid(xrange(1):step:xrange(2),...
                   yrange(1):step:yrange(2));
  Z = sin(X).*cos(Y);
  surf(X,Y,Z); colormap(jet); colorbar;
  xlabel('x'); ylabel('y'); zlabel('z');
end
```

Listing 1.5: Sobota řešení — úkoly 1–20

1.6.3 Projekt: Univerzální skript pro vizualizaci $z = \sin(x)\cos(y)$

```
% Projekt: Vizualizace funkce z = sin(x)*cos(y)

% 1) Nastavení parametrů
xrange = [-5 5];        % rozsah osy X
yrange = [-5 5];        % rozsah osy Y
step = 0.1;             % hustota vzorkování

% 2) Vytvoření 2D sítě
[x,y] = meshgrid(xrange(1):step:xrange(2), ...
                 yrange(1):step:yrange(2));

% 3) Definice funkce
z = sin(x).*cos(y);

% 4) Vykreslení 3D povrchu
figure;
surf(x,y,z);

% 5) Nastavení grafu
colormap(jet);          % barevná mapa
colorbar;               % barevná škála
shading interp;         % vyhlazení povrchu
alpha(0.9);             % průhlednost
xlabel('X osa'); ylabel('Y osa'); zlabel('Z = sin(x)cos(y)');
title('Vizualizace 2D funkce z = sin(x)cos(y)');

% 6) Úprava kamery
view(45,30);            % natočení grafu
axis tight;             % oseknutí na data
grid on;                % mřížka

% 7) Export do souboru
saveas(gcf,'projekt_sin_cos.png');
```

Listing 1.6: Kompletní projekt s detailním vysvětlením

1.6.4 Vysvětlení kódu

1. `xrange`, `yrange`, `step` – určují, jakou část roviny XY a s jakou hustotou zobrazíme.
2. `meshgrid` – převádí 1D vektory x, y na 2D mřížky souřadnic (X, Y).
3. `z = sin(x).*cos(y)` – element-wise násobení, nutné kvůli práci s maticemi.
4. `surf` – kreslí 3D povrch.
5. `colormap`, `colorbar`, `shading`, `alpha` – ovládají vizuální styl.
6. `view(45,30)` – nastavuje perspektivu kamery.
7. `saveas` – ukládá výsledek do obrázku (PNG).

1.7 Neděle: Rekapitulace týdne

1.7.1 Procvičovací úkoly

1. Vytvořte vektor $v = [1 : 10]$ a zobrazte ho.
2. Vytvořte matici 3×3 s čísly 1–9 a vypište prvek na pozici (2,3).
3. Sečtěte dva vektory $a = [1, 2, 3]$, $b = [4, 5, 6]$.
4. Vynásobte dvě matice $A = eye(2)$ a $B = [2, 3; 4, 5]$.
5. Vypočítejte inverzi matice $[2, 1; 1, 2]$.
6. Vytvořte vektor $x = -2 : 0.1 : 2$ a spočítejte $y = x.^2$.
7. Vykreslete graf $y = x.^2$ pomocí `plot`.
8. Vykreslete v jednom grafu funkce $y1 = x$, $y2 = x.^2$, $y3 = x.^3$.
9. Přidejte k předchozímu grafu legendu (`legend`).
10. Vytvořte vektor $t = 0 : 0.1 : 10$ a vykreslete $sin(t)$ červeně a $cos(t)$ modře.
11. Vytvořte matici náhodných čísel 3×3 pomocí `rand` a zobrazte ji.
12. Vypočítejte determinant matice $[1, 2, 3; 4, 5, 6; 7, 8, 10]$.
13. Vytvořte 3D síť pro $x, y \in [-3, 3]$ s krokem 0.1 a vykreslete $z = x.^2 + y.^2$.
14. Vykreslete funkci $z = sin(x). * cos(y)$ pomocí `surf`.
15. Upravte hustotu sítě předchozího grafu na krok 0.05 a sledujte rozdíl.
16. Přidejte k grafu $z = sin(x). * cos(y)$ popisky os a nadpis.
17. Vytvořte konturový graf $z = sin(x). * cos(y)$ pomocí `contour`.
18. Uložte poslední obrázek do souboru PNG (`saveas`).
19. Napište skript, který pro daný rozsah a krok vykreslí 3D povrch $z = sin(x). * cos(y)$.
20. **Vlastní příklad:** Vytvořte vektor $x = -10 : 0.1 : 10$, spočítejte $y = \sin(x)./x$ (s podmínkou, že pro $x = 0$ nastavíte $y = 1$), a vykreslete graf.

1.7.2 Poznámky k novým příkazům

- `legend` — umožňuje přidat do grafu popisy jednotlivých křivek. Syntaxe: `legend('popisek1','pop`
- `rand` — generuje náhodná čísla z intervalu (0,1). Např. `rand(3,3)` vytvoří 3×3 matici náhodných čísel.

1.7.3 Řešení rekapitulace 1. týdne

```
%% 1. Vytvoření vektoru v
v = 1:10;

%% 2. Matice 3x3 a prvek (2,3)
A = reshape(1:9,3,3);
elem_23 = A(2,3);

%% 3. Součet vektorů
a = [1,2,3]; b = [4,5,6];
sum_vec = a + b;

%% 4. Maticové násobení
B = [2,3;4,5];
C = eye(2) * B;

%% 5. Inverze matice
M = [2,1;1,2];
M_inv = inv(M);

%% 6. Kvadratická funkce
x = -2:0.1:2;
y = x.^2;

%% 7. Plot y=x.^2
figure; plot(x,y);

%% 8. Více funkcí v jednom grafu
y1 = x; y2 = x.^2; y3 = x.^3;
figure; plot(x,y1,'r',x,y2,'g',x,y3,'b');

%% 9. Přidání legendy
legend('y=x','y=x^2','y=x^3');

%% 10. Sin a cos v jednom grafu
t = 0:0.1:10;
figure; plot(t,sin(t),'r', t,cos(t),'b');

%% 11. Náhodná matice 3x3
R = rand(3,3);

%% 12. Determinant matice
detA = det([1,2,3;4,5,6;7,8,10]);

%% 13. 3D síť a paraboloid
[x,y] = meshgrid(-3:0.1:3, -3:0.1:3);
z = x.^2 + y.^2;
figure; surf(x,y,z);

%% 14. Funkce z = sin(x).*cos(y)
```

```
[X,Y] = meshgrid(-3:0.1:3,-3:0.1:3);
Z = sin(X).*cos(Y);
figure; surf(X,Y,Z);

%% 15. Hustší síť
[X2,Y2] = meshgrid(-3:0.05:3, -3:0.05:3);
Z2 = sin(X2).*cos(Y2);
figure; surf(X2,Y2,Z2);

%% 16. Popisky a nadpis
xlabel('x'); ylabel('y'); zlabel('z');
title('z = sin(x)cos(y)');

%% 17. Konturový graf
figure; contour(X,Y,Z);

%% 18. Uložení obrázku do PNG
saveas(gcf,'rekapitulace_z.png');

%% 19. Skript pro libovolný rozsah a krok
% Funkce je v samostatném souboru nebo v příkazovém okně
function plotSineCos(xrange,yrange,step)
    [X,Y] = meshgrid(xrange(1):step:xrange(2), ...
                     yrange(1):step:yrange(2));
    Z = sin(X).*cos(Y);
    surf(X,Y,Z); colormap(jet); colorbar;
    xlabel('x'); ylabel('y'); zlabel('z');
end

%% 20. Vlastní příklad: sinc funkce
x = -10:0.1:10;
y = zeros(size(x));
y(x~=0) = sin(x(x~=0))./x(x~=0);
y(x==0) = 1;
figure; plot(x,y);
xlabel('x'); ylabel('y = sin(x)/x');
title('Sinc funkce');
grid on;
```

Listing 1.7: Neděle: Rekapitulace týdne — kompletní řešení

1.7.4 Vysvětlení kódu

1. **Vektory a matice:** Vytváříme je standardním způsobem `1:10`, `reshape`.
2. **Indexování:** Pro prvek (2,3) používáme `A(2,3)`.
3. **Součet vektorů a maticové násobení:** `+` pro element-wise, `*` pro maticové.
4. **Inverze:** `inv(M)`.
5. **Element-wise operace:** *.promocninypoprvcích*, *.*pronásobenípoprvcích*.**Grafy:** `plot`, `surf`, `con`

6. **Náhodná matice:** `rand(3,3)`.
7. **Determinant:** `det(A)`.
8. **Ukládání obrázků:** `saveas(gcf,'soubor.png')`.
9. **Nové příkazy:**
   - `legend` – přidává popisky křivek.
   - `rand` – generuje náhodná čísla (0,1).
   - `grid on` – zobrazuje mřížku v grafu.
   - Vlastní příklad: `y(x ==0) = sin(x(x ==0))./x(x ==0);` – podmíněné přiřazení prvků pro x0.

Kapitola 2

2 Týden - Funkce a skripty

2.1 Pondělí: Rozdíl skript vs. funkce, první vlastní funkce

2.1.1 Úkoly

1. Vytvořte jednoduchý skript, který vypíše "Ahoj svět".
2. Vytvořte skript, který sečte čísla 1 až 10.
3. Vytvořte skript, který vytvoří vektor $v = 1 : 5$ a vypíše jeho průměr.
4. Napište skript, který vykreslí funkci $y = x^2$ pro $x = -5 : 0.1 : 5$.
5. Vytvořte skript, který vytvoří matici 3×3 náhodných čísel a vypíše její determinant.
6. Napište jednoduchou funkci, která vrátí součet dvou čísel.
7. Napište funkci, která vrátí více výstupů: součet a rozdíl dvou čísel.
8. Napište funkci s parametrem, který určuje exponent v $y = x^n$.
9. Napište funkci, která vrátí průměr a směrodatnou odchylku vektoru.
10. Vytvořte funkci, která vypíše všechny prvky matice větší než průměr.
11. Napište funkci, která pro vstupní vektor vrátí vektor druhých mocnin.
12. Vytvořte skript, který zavolá funkci z předchozího úkolu a vykreslí výsledky.
13. Napište funkci, která ověří, zda je matice čtvercová.
14. Napište funkci, která vrátí první n Fibonacciho čísla.
15. Napište funkci, která vrátí faktoriál zadaného čísla.
16. Vytvořte funkci, která převrátí pořadí vektoru.
17. Napište funkci, která spočítá RMS hodnotu vektoru.
18. Vytvořte funkci, která zjistí maximum a minimum vektoru.
19. Napište skript, který použije několik z předchozích funkcí a vypíše výsledky.
20. Vytvořte vlastní příklad: skript, který vytvoří matici 5×5 s čísly 1–25 a použije funkci pro součet řádků.

2.1.2 Řešení (kompaktní MATLAB kód)

```
%% 1. Skript "Ahoj svět"
disp('Ahoj svět');

%% 2. Skript: součet čísel 1 až 10
sum1to10 = sum(1:10);

%% 3. Skript: průměr vektoru
v = 1:5;
mean_v = mean(v);

%% 4. Skript: vykreslení y = x^2
x = -5:0.1:5;
y = x.^2;
figure; plot(x,y);

%% 5. Skript: determinant náhodné matice
A = rand(3,3);
detA = det(A);

%% 6. Funkce: součet dvou čísel
% function s = sumTwo(a,b)
%     s = a + b;
% end

%% 7. Funkce: součet a rozdíl
% function [s,r] = sumDiff(a,b)
%     s = a + b;
%     r = a - b;
% end

%% 8. Funkce: exponent y = x^n
% function y = powerFunc(x,n)
%     y = x.^n;
% end

%% 9. Funkce: průměr a odchylka
% function [m,s] = meanStd(v)
%     m = mean(v);
%     s = std(v);
% end

%% 10. Funkce: prvky větší než průměr
% function idx = greaterThanMean(A)
%     idx = A(A > mean(A(:)));
% end

%% 11. Funkce: druhé mocniny
% function y2 = squareElements(v)
%     y2 = v.^2;
```

```
% end

%% 12. Skript: volání funkce squareElements
v2 = 1:10;
v2_squared = squareElements(v2);
figure; plot(v2,v2_squared);

%% 13. Funkce: ověření čtvercové matice
% function flag = isSquare(A)
%     [m,n] = size(A);
%     flag = (m == n);
% end

%% 14. Funkce: Fibonacci
% function F = fibonacci(n)
%     F = zeros(1,n);
%     F(1) = 1; F(2) = 1;
%     for k = 3:n
%         F(k) = F(k-1) + F(k-2);
%     end
% end

%% 15. Funkce: faktoriál
% function f = factorialFunc(n)
%     f = prod(1:n);
% end

%% 16. Funkce: převrácení vektoru
% function vrev = reverseVector(v)
%     vrev = v(end:-1:1);
% end

%% 17. Funkce: RMS
% function r = rmsVec(v)
%     r = sqrt(mean(v.^2));
% end

%% 18. Funkce: max a min
% function [mx,mn] = maxMin(v)
%     mx = max(v);
%     mn = min(v);
% end

%% 19. Skript: kombinace funkcí
a = 3; b = 5;
[s,r] = sumDiff(a,b);
f5 = factorialFunc(5);

%% 20. Vlastní příklad: součet řádků matice 5x5
M = reshape(1:25,5,5);
rowSums = sum(M,2);
```

Listing 2.1: Pondělí: Skript vs. funkce — kompletní řešení

2.1.3 Řešení s podrobným vysvětlením kódu

1. **Skript "Ahoj svět"**: `disp('Ahoj svět')` – jednoduchý příkaz k vypsání textu.
2. **Součet čísel 1–10**: `sum(1:10)` – součet prvků vektoru.
3. **Průměr vektoru**: `mean(v)` – průměr aritmetický.
4. **Graf funkce**: `x.^2` *element−wisemocnina*; `plot` *vykreslígraf*. **`Determinant náhodné matice`** : `rand(3,3)` *generuje*3*x*3*matice*; `det` *vypočítádeterminant*.
5. **Funkce součet dvou čísel**: standardní `function s = sumTwo(a,b)`; návratová hodnota.
6. **Součet a rozdíl**: více výstupů `[s,r]`.
7. **Exponent y=x**^{n} : *parametr* **n** *určujemocninu*; *element* − *wiseoperace*

2.1.4 Poznámky k novým příkazům a konceptům

- `function` – definice funkce, první řádek určuje název, vstupy a výstupy.
- Více výstupů – `[out1,out2] = func(...)`.
- Element-wise operátory: `.`, `.*`, `./`. *Maskovánímatic* : *výběrprvků splňujících podmínku*.

2.2 Úterý: Parametry funkcí a více výstupů

2.2.1 Úkoly

1. Napište funkci, která vrátí součet tří čísel.
2. Vytvořte funkci, která má dva vstupy (strany obdélníka) a vrátí jeho obsah.
3. Napište funkci, která má dva vstupy (strany obdélníka) a vrátí obsah i obvod.
4. Vytvořte funkci, která pro číslo n vrátí vektor $1, 2, \ldots, n$.
5. Napište funkci, která pro číslo n vrátí součet $1 + 2 + \cdots + n$.
6. Vytvořte funkci, která vrátí součet, průměr a maximum vektoru.
7. Napište funkci, která vrátí skalární součin dvou vektorů.
8. Napište funkci, která pro matici vrátí počet řádků i počet sloupců.
9. Napište funkci, která pro vstupní vektor vrátí vektor s druhými a třetími mocninami (dva výstupy).
10. Vytvořte funkci, která přijme a, b a vrátí a^b i b^a.
11. Napište funkci, která pro číslo n vrátí faktoriál a Fibonacciho číslo.
12. Vytvořte funkci, která pro vektor vrátí medián a rozptyl.
13. Napište funkci, která vrátí součet lichých a součet sudých čísel od 1 do n.
14. Vytvořte funkci, která přijme 3D vektor a vrátí jeho velikost a normalizovanou verzi.
15. Napište funkci, která přijme matici a vrátí součet všech řádků a všech sloupců (dva výstupy).
16. Napište funkci, která pro a, b, c vrátí kořeny kvadratické rovnice.
17. Vytvořte funkci, která má proměnný počet vstupů a vrátí jejich součet (nový koncept).
18. Napište funkci, která vrátí průměr hodnot a počet hodnot větších než průměr.
19. Vytvořte funkci, která vrátí minimální a maximální prvek matice i jejich pozici.
20. Vlastní příklad: funkce, která spočítá součet čísel od m do n a jejich průměr.

2.2.2 Řešení (kompaktní MATLAB kód)

```
%% 1. Součet tří čísel
% function s = sumThree(a,b,c)
%     s = a + b + c;
% end

%% 2. Obsah obdélníka
% function A = areaRect(a,b)
%     A = a*b;
% end

%% 3. Obsah a obvod obdélníka
% function [A,P] = rectGeom(a,b)
%     A = a*b;
%     P = 2*(a+b);
% end

%% 4. Vektor 1:n
% function v = oneToN(n)
%     v = 1:n;
% end

%% 5. Součet 1+...+n
% function s = sumN(n)
%     s = sum(1:n);
% end

%% 6. Součet, průměr, maximum
% function [s,mx,av] = statsVec(v)
%     s = sum(v);
%     mx = max(v);
%     av = mean(v);
% end

%% 7. Skalární součin
% function s = dotProd(u,v)
%     s = sum(u.*v);
% end

%% 8. Rozměry matice
% function [r,c] = matrixSize(A)
%     [r,c] = size(A);
% end

%% 9. Druhé a třetí mocniny
% function [v2,v3] = powers(v)
%     v2 = v.^2;
%     v3 = v.^3;
% end

```

```
%% 10. a^b a b^a
% function [ab,ba] = powerBoth(a,b)
%     ab = a^b;
%     ba = b^a;
% end

%% 11. Faktoriál a Fibonacci
% function [f,F] = factFib(n)
%     f = prod(1:n);
%     F = zeros(1,n); F(1)=1;F(2)=1;
%     for k=3:n
%         F(k)=F(k-1)+F(k-2);
%     end
% end

%% 12. Medián a rozptyl
% function [med,varv] = medVar(v)
%     med = median(v);
%     varv = var(v);
% end

%% 13. Součet lichých a sudých
% function [oddSum,evenSum] = oddEvenSum(n)
%     oddSum = sum(1:2:n);
%     evenSum = sum(2:2:n);
% end

%% 14. Velikost a normalizace vektoru
% function [len,u] = normalize3(v)
%     len = norm(v);
%     u = v/len;
% end

%% 15. Součet řádků a sloupců
% function [rowSums,colSums] = sumRowsCols(A)
%     rowSums = sum(A,2);
%     colSums = sum(A,1);
% end

%% 16. Kvadratická rovnice
% function roots = quadRoots(a,b,c)
%     D = b^2-4*a*c;
%     roots = [(-b+sqrt(D))/(2*a), (-b-sqrt(D))/(2*a)];
% end

%% 17. Proměnný počet vstupů (nové)
% function s = sumVarargin(varargin)
%     s = sum([varargin{:}]);
% end

%% 18. Průměr a počet nadprůměrných
```

```
% function [avg,count] = avgAbove(v)
%     avg = mean(v);
%     count = sum(v > avg);
% end

%% 19. Min, max a pozice
% function [mn,mx,posMn,posMx] = minMaxPos(A)
%     [mx,posMx] = max(A(:));
%     [mn,posMn] = min(A(:));
% end

%% 20. Vlastní příklad: součet a průměr od m do n
% function [s,av] = sumAvg(m,n)
%     s = sum(m:n);
%     av = mean(m:n);
% end
```

Listing 2.2: Úterý: Parametry funkcí a více výstupů — řešení

2.2.3 Řešení s podrobným vysvětlením kódu

1. **Součet tří čísel** – funkce přijímá tři vstupy, sčítá je a vrací výsledek.
2. **Obsah obdélníka** – jednoduchý součin stran.
3. **Obsah a obvod** – funkce s *více výstupy*.
4. **Vektor 1:n** – generace aritmetické posloupnosti.
5. **Součet 1...n** – využití `sum`.
6. **Statistiky vektoru** – kombinace `sum`, `max`, `mean`.
7. **Skalární součin** – definován jako $\sum u_i v_i$.
8. **Rozměry matice** – použití `size`.
9. **Druhé a třetí mocniny** – dva výstupy vracející různé transformace vektoru.
10. **a na b a b na a** – ukázka dvou výsledků z jedné funkce.
11. **Faktoriál a Fibonacci** – kombinace dvou výpočtů.
12. **Medián a rozptyl** – použití vestavěných funkcí `median`, `var`.
13. **Součet lichých a sudých čísel** – generování posloupností a součet.
14. **Velikost a normalizace** – využití `norm`, pak dělení vektoru délkou.
15. **Součet řádků a sloupců** – `sum(A,2)` řádky, `sum(A,1)` sloupce.
16. **Kvadratická rovnice** – klasický vzorec, dva kořeny.
17. **Proměnný počet vstupů** – novinka: `varargin`, pole všech argumentů.
18. **Průměr a nadprůměrné hodnoty** – kombinace `mean` a logické podmínky.
19. **Min, max a pozice** – funkce `min`, `max` s indexy na linearizovaném poli.
20. **Součet a průměr od m do n** – vlastní příklad kombinující sumu a průměr.

2.2.4 Nové koncepty

- Více výstupů z funkce: `[out1,out2] = f(...)`.
- Proměnný počet vstupů: `varargin` (pole všech argumentů).
- Funkce `norm`, `median`, `var`.
- Vrácení hodnot i indexů pomocí `min`, `max`.

2.3 Středa: Funkce a práce s vektory/maticemi

2.3.1 Úkoly

1. Funkce, která vrátí délku vektoru.
2. Funkce, která vrátí součet všech prvků vektoru.
3. Funkce, která vrátí maximum a minimum vektoru.
4. Funkce, která vrátí průměr prvků matice.
5. Funkce, která vrátí transponovanou matici.
6. Funkce, která vrátí diagonálu matice jako vektor.
7. Funkce, která vrátí součet prvků na hlavní diagonále.
8. Funkce, která vrátí horní trojúhelníkovou část matice.
9. Funkce, která vrátí dolní trojúhelníkovou část matice.
10. Funkce, která vrátí inverzi prvků vektoru (tj. $1/x$).
11. Funkce, která vrátí normalizovaný vektor (součet prvků = 1).
12. Funkce, která vrátí součin matice a vektoru.
13. Funkce, která vrátí matici otočenou o 90°.
14. Funkce, která vrátí řádkové a sloupcové součty matice.
15. Funkce, která zjistí, zda je matice symetrická.
16. Funkce, která pro vektor vrátí histogram (počty výskytů hodnot).
17. Funkce, která vrátí matici bez prvního a posledního řádku.
18. Funkce, která pro vektor vrátí jeho seřazení vzestupně i sestupně.
19. Funkce, která vrátí Frobeniovu normu matice.
20. Vlastní příklad: funkce, která vrátí rozdíl mezi největším a nejmenším prvkem matice.

2.3.2 Řešení (kompaktní MATLAB kód)

```
%% 1. Délka vektoru
% function l = vecLen(v)
%     l = length(v);
% end

%% 2. Součet prvků vektoru
% function s = sumVec(v)
%     s = sum(v);
% end

%% 3. Max a min vektoru
% function [mx,mn] = minMaxVec(v)
%     mx = max(v);
%     mn = min(v);
% end

%% 4. Průměr matice
% function m = meanMat(A)
%     m = mean(A(:));
% end

%% 5. Transpozice
% function B = transposeMat(A)
%     B = A';
% end

%% 6. Diagonála
% function d = diagVec(A)
%     d = diag(A);
% end

%% 7. Součet diagonály
% function s = sumDiag(A)
%     s = sum(diag(A));
% end

%% 8. Horní trojúhelník
% function U = upperTri(A)
%     U = triu(A);
% end

%% 9. Dolní trojúhelník
% function L = lowerTri(A)
%     L = tril(A);
% end

%% 10. Inverze prvků vektoru
% function invv = invVec(v)
%     invv = 1./v;
```

```
% end

%% 11. Normalizace vektoru (součet=1)
% function vn = normVec(v)
%     vn = v/sum(v);
% end

%% 12. Součin matice a vektoru
% function u = matVec(A,v)
%     u = A*v;
% end

%% 13. Rotace o 90
% function R = rot90Mat(A)
%     R = rot90(A);
% end

%% 14. Součet řádků a sloupců
% function [r,c] = sumRC(A)
%     r = sum(A,2);
%     c = sum(A,1);
% end

%% 15. Symetrie matice
% function tf = isSym(A)
%     tf = isequal(A,A');
% end

%% 16. Histogram výskytů
% function h = histCount(v)
%     vals = unique(v);
%     h = [vals' histc(v,vals)'];
% end

%% 17. Bez prvního a posledního řádku
% function B = cutRows(A)
%     B = A(2:end-1,:);
% end

%% 18. Seřazení vzestupně a sestupně
% function [asc,desc] = sortBoth(v)
%     asc = sort(v);
%     desc = sort(v,'descend');
% end

%% 19. Frobeniova norma
% function f = frobNorm(A)
%     f = norm(A,'fro');
% end

%% 20. Rozdíl max - min
```

```
% function d = rangeMat(A)
%     d = max(A(:))-min(A(:));
% end
```

Listing 2.3: Středa: Funkce a práce s vektory/maticemi — řešení

2.3.3 Řešení s podrobným vysvětlením kódu

1. **Délka vektoru** – funkce `length` vrací počet prvků.
2. **Součet prvků** – `sum` sčítá všechny prvky vektoru.
3. **Max a min** – kombinace `max`, `min`.
4. **Průměr matice** – `A(:)` rozvine matici na vektor, pak `mean`.
5. **Transpozice** – operátor `'` převrátí řádky a sloupce.
6. **Diagonála** – funkce `diag` extrahuje hlavní diagonálu.
7. **Součet diagonály** – `sum(diag(A))`.
8. **Horní/dolní trojúhelník** – funkce `triu`, `tril`.
9. **Inverze prvků vektoru** – operace `1./v` je po složkách.
10. **Normalizace vektoru** – vydělení součtem všech prvků.
11. **Součin matice a vektoru** – klasická lineární algebra `A*v`.
12. **Rotace o 90°** – funkce `rot90`.
13. **Součty řádků a sloupců** – `sum(A,2)` řádky, `sum(A,1)` sloupce.
14. **Symetrie** – matice je symetrická, když $A = A^T$.
15. **Histogram** – `unique` najde unikátní hodnoty, `histc` spočítá četnosti.
16. **Oříznutí matice** – indexování $A(2 : end - 1, :)$.
17. **Seřazení** – `sort`, možnost parametru `'descend'`.
18. **Frobeniova norma** – $\|A\|_F = \sqrt{\sum a_{ij}^2}$, funkce `norm(A,'fro')`.
19. **Rozsah matice** – rozdíl maxima a minima z `A(:)`.

2.3.4 Nové koncepty

- Funkce `diag`, `triu`, `tril`, `rot90`.
- Histogram hodnot pomocí `unique`, `histc`.
- Frobeniova norma: `norm(A,'fro')`.

2.4 Čtvrtek: Funkce a podmínky (if, else, switch)

2.4.1 Úkoly

1. Funkce, která zjistí, zda je číslo kladné nebo záporné.
2. Funkce, která zjistí, zda je číslo sudé nebo liché.
3. Funkce, která zjistí, zda je číslo dělitelné 3.
4. Funkce, která vrátí absolutní hodnotu čísla (bez použití abs).
5. Funkce, která vrátí největší ze dvou čísel.
6. Funkce, která vrátí největší ze tří čísel.
7. Funkce, která zjistí, zda má vektor délku větší než 5.
8. Funkce, která zjistí, zda je matice čtvercová.
9. Funkce, která vrátí počet kladných čísel ve vektoru.
10. Funkce, která rozdělí číslo na intervaly: < 0, $0 - 10$, > 10.
11. Funkce, která při vstupu 1–7 vrátí jméno dne v týdnu (switch).
12. Funkce, která zjistí, zda je rok přestupný.
13. Funkce, která určí známku z bodů (90+ = A, 80+ = B, jinak F).
14. Funkce, která zjistí, zda je matice diagonální.
15. Funkce, která vrátí "ano"/"ne"podle pravdivosti logického výrazu.
16. Funkce, která porovná dva vektory na rovnost.
17. Funkce, která zjistí, zda je prvek přítomen ve vektoru.
18. Funkce, která vrátí typ čísla: záporné, nula, kladné.
19. Funkce, která klasifikuje úhel (ostrý, pravý, tupý).
20. Vlastní příklad: funkce, která určí, zda matice obsahuje jen nenulové prvky.

2.4.2 Řešení (kompaktní MATLAB kód)

```
%% 1. Kladné nebo záporné
% function s = signCheck(x)
%   if x>0, s='kladné';
%   elseif x<0, s='záporné';
%   else, s='nula'; end
% end

%% 2. Sudé/liché
% function s = evenOdd(x)
```

```
%   if mod(x,2)==0, s='sudé'; else, s='liché'; end
% end

%% 3. Dělitelné 3
% function tf = div3(x)
%   tf = (mod(x,3)==0);
% end

%% 4. Absolutní hodnota
% function y = absOwn(x)
%   if x<0, y=-x; else, y=x; end
% end

%% 5. Maximum ze 2 čísel
% function m = max2(a,b)
%   if a>b, m=a; else, m=b; end
% end

%% 6. Maximum ze 3 čísel
% function m = max3(a,b,c)
%   if a>=b && a>=c, m=a;
%   elseif b>=a && b>=c, m=b;
%   else, m=c; end
% end

%% 7. Vektor délky >5
% function tf = longVec(v)
%   if length(v)>5, tf=true; else, tf=false; end
% end

%% 8. Čtvercová matice?
% function tf = isSquare(A)
%   [m,n] = size(A);
%   tf = (m==n);
% end

%% 9. Počet kladných čísel
% function c = countPos(v)
%   c = sum(v>0);
% end

%% 10. Interval čísel
% function out = intervalCheck(x)
%   if x<0, out='záporné';
%   elseif x<=10, out='0-10';
%   else, out='větší než 10'; end
% end

%% 11. Den v týdnu (switch)
% function d = dayName(n)
%   switch n
```

```
%      case 1, d='Pondělí';
%      case 2, d='Úterý';
%      case 3, d='Středa';
%      case 4, d='Čtvrtek';
%      case 5, d='Pátek';
%      case 6, d='Sobota';
%      case 7, d='Neděle';
%      otherwise, d='Neplatné číslo';
%    end
% end

%% 12. Přestupný rok
% function tf = leapYear(r)
%    if mod(r,400)==0 || (mod(r,4)==0 && mod(r,100)~=0)
%        tf=true;
%    else, tf=false; end
% end

%% 13. Známka
% function z = grade(p)
%    if p>=90, z='A';
%    elseif p>=80, z='B';
%    else, z='F'; end
% end

%% 14. Diagonální matice?
% function tf = isDiag(A)
%    tf = isequal(A,diag(diag(A)));
% end

%% 15. Ano/Ne pro logický výraz
% function out = yesNo(cond)
%    if cond, out='ano'; else, out='ne'; end
% end

%% 16. Rovnost vektorů
% function tf = eqVec(a,b)
%    tf = isequal(a,b);
% end

%% 17. Přítomnost prvku
% function tf = inVec(v,x)
%    tf = any(v==x);
% end

%% 18. Typ čísla
% function t = numType(x)
%    if x<0, t='záporné';
%    elseif x==0, t='nula';
%    else, t='kladné'; end
% end
```

```
112
113 %% 19. Klasifikace úhlu
114 % function t = angleType(a)
115 %   if a<90, t='ostrý';
116 %   elseif a==90, t='pravý';
117 %   else, t='tupý'; end
118 % end
119
120 %% 20. Nenulová matice?
121 % function tf = nonZeroMat(A)
122 %   tf = all(A(:)~=0);
123 % end
```

Listing 2.4: Čtvrtek: Funkce a podmínky — řešení

2.4.3 Řešení s podrobným vysvětlením

1. `if/elseif/else` rozhoduje podle znaménka čísla.
2. `mod(x,2)` zbytek po dělení určuje sudost/lichost.
3. Dělitelnost 3 pomocí `mod(x,3)`.
4. Vlastní absolutní hodnota: negujeme jen záporné číslo.
5. –6. Porovnávání více čísel podmínkami.
6. Kontrola délky vektoru přes `length`.
7. Čtvercová matice: stejné počty řádků a sloupců.
8. Počet kladných čísel: logické porovnání a `sum`.
9. Rozdělení na intervaly pomocí vnořených podmínek.
10. `switch` – alternativa k if–elseif–else, vhodné pro diskrétní hodnoty.
11. Přestupný rok: pravidla dělitelnosti 400, 100, 4.
12. Známkování: více úrovní podmínek.
13. Diagonální matice má všechny prvky mimo diagonálu nulové.
14. Podmínka jako ano/ne (logická hodnota do textu).
15. Porovnání vektorů: funkce `isequal`.
16. Přítomnost prvku: `any(v==x)`.
17. Rozdělení čísla na záporné, nulu, kladné.
18. Úhly: klasifikace podle stupňů.
19. Nenulová matice: test `all(A(:) =0)`.

2.4.4 Nové koncepty

- `if`, `elseif`, `else` – základní podmínky.
- `switch-case` – větvení podle diskrétních hodnot.
- Logické funkce: `any`, `all`, `isequal`.

2.5 Pátek: Lokální a globální proměnné, struktury, cell arrays, varargin/varargout, ukládání dat

2.5.1 Úkoly

1. Ukaž rozdíl mezi lokální a globální proměnnou.
2. Funkce, která používá globální proměnnou pro uchování počítadla.
3. Vytvoř strukturu se jménem, věkem a výškou.
4. Přístup k poli ve struktuře (vypsat jen věk).
5. Vytvoř pole struktur pro více studentů.
6. Funkce, která vrátí průměrný věk ze struktury studentů.
7. Vytvoř cell array s číslem, textem a vektorem.
8. Přístup k prvku cell array (druhý prvek).
9. Převod cell array na klasické pole (pokud obsahuje čísla).
10. Vytvoř vnořenou cell array (cell uvnitř cell).
11. Funkce, která sečte libovolný počet vstupů (varargin).
12. Funkce, která vrátí minimum a maximum současně (varargout).
13. Funkce, která zpracuje text nebo číslo podle vstupu (varargin).
14. Vytvoř strukturu s vektorem a proveď aritmetiku s prvky.
15. Ulož data do .mat souboru a znovu je načti.
16. Ulož data do .txt souboru (save -ascii).
17. Načti textový soubor zpět do MATLABu.
18. Funkce, která ukládá výsledky výpočtu do souboru s automatickým názvem.
19. Funkce, která pracuje se strukturou obsahující cell array.
20. Vlastní příklad: Funkce, která načte data, provede výpočet a uloží do nové struktury.

2.5.2 Řešení (kompaktní MATLAB kód)

```
%% 1. Lokální vs globální
% function f1
%    a = 5; % lokální
%    global b; b = 10; % globální
% end

%% 2. Globální počítadlo
% function counter
%    global c
%    if isempty(c), c=0; end
%    c = c+1
% end

%% 3. Struktura osoba
osoba.jmeno = 'Jan'; osoba.vek = 25; osoba.vyska = 180;

%% 4. Přístup k poli
vek = osoba.vek;

%% 5. Pole struktur
student(1).jmeno='Eva'; student(1).vek=20;
student(2).jmeno='Petr'; student(2).vek=22;

%% 6. Průměrný věk
% function prum = avgAge(stud)
%    veky = [stud.vek];
%    prum = mean(veky);
% end

%% 7. Cell array
C = {42,'text',[1 2 3]};

%% 8. Přístup k cell
druhy = C{2};

%% 9. Převod na pole
nums = cell2mat({1,2,3});

%% 10. Vnořená cell
C2 = { {1,2}, 'Ahoj' };

%% 11. Libovolný počet vstupů
% function s = sumVarargin(varargin)
%    s=0;
%    for k=1:nargin, s=s+varargin{k}; end
% end

%% 12. Varargout
% function [varargout] = minmaxVec(v)
```

```
%    varargout{1}=min(v);
%    varargout{2}=max(v);
% end

%% 13. Text nebo číslo
% function out = process(varargin)
%    if ischar(varargin{1}), out=upper(varargin{1});
%    else, out=varargin{1}^2; end
% end

%% 14. Struktura s vektorem
data.vec = [1 2 3];
soucin = prod(data.vec);

%% 15. Uložení do MAT
save('osoba.mat','osoba')
load('osoba.mat')

%% 16. Uložení do TXT
M = [1 2;3 4];
save('matice.txt','M','-ascii')

%% 17. Načtení TXT
X = load('matice.txt');

%% 18. Ukládání s názvem
% function saveRes(x)
%    fname = ['res_' datestr(now,'HHMMSS') '.mat'];
%    save(fname,'x');
% end

%% 19. Struktura s cell
S.celldata = {1:3,'abc'};
out1 = S.celldata{1}(2);

%% 20. Vlastní příklad
% function S = processData(v)
%    avg=mean(v); sumv=sum(v);
%    S.vstup=v; S.prumer=avg; S.soucet=sumv;
%    save('results.mat','S');
% end
```

Listing 2.5: Pátek – řešení

2.5.3 Řešení s podrobným vysvětlením

1. **Lokální proměnné** existují jen uvnitř funkce; **globální** jsou sdílené napříč workspace (definované příkazem `global`).
2. Globální proměnná může uchovávat stav (počítadlo), které přetrvává mezi voláními.
3. –6. Struktura je datový typ, který pojí jména polí s hodnotami; lze z ní tvořit pole struktur a počítat statistiky.
4. –10. Cell array umožňuje uchovávat heterogenní data (čísla, text, vektory, jiné cell).
5. `varargin` – funkce může přijmout libovolný počet vstupů (uložených do cell pole).
6. `varargout` – funkce může vrátit libovolný počet výstupů.
7. Ukládání dat: `save/load` pro `.mat` soubory, `-ascii` pro textové soubory.
8. Funkce může automaticky generovat název výstupního souboru pomocí `datestr(now)`.
9. Struktury a cell lze kombinovat (struktura obsahuje cell, cell obsahuje vektor atd.).
10. Vlastní příklad ukazuje spojení všech technik: načtení vstupu, výpočty, uložení do struktury a export do souboru.

2.5.4 Nové koncepty

- `global` – globální proměnná.
- Struktury: `S.field`, pole struktur.
- Cell arrays: `C{i}`, `cell2mat`.
- `varargin`, `varargout`.
- Ukládání: `save`, `load`, `datestr`.

2.6 Sobota: Cvičení a projekt — Struktury, cell arrays, varargin/varargout, ukládání, CSV a grafy

2.6.1 Úkoly (20)

1. Vytvořte strukturu `person` s poli `name`, `age`, `tempHistory` (vektor).
2. Přidejte do struktury nové pole `location` a přiřaďte textovou hodnotu.
3. Vytvořte pole struktur `persons` pro tři osoby, každé s jinou historií teplot.
4. Vypočítejte průměrnou teplotu pro každou osobu a uložte do pole `avgTemp` ve struktuře.
5. Vytvořte `cell` pole `C` obsahující: číslo, text a vektor hodnot teplot.
6. Zkopírujte druhý prvek cell pole do proměnné jako řetězec.
7. Pokud jsou v cell poli pouze číselné vektory, spojte je do jedné matice pomocí `cell2mat`.
8. Vytvořte vnořenou strukturu: `station.info.name` a `station.data.temps`.
9. Uložte strukturu `persons` do souboru `persons.mat` a znovu ji načtěte.
10. Exportujte průměry všech osob do textového souboru `avgTemps.txt` (ASCII).
11. Vytvořte funkci `saveWithTimestamp(x)` — uloží proměnnou `x` do MAT se jménem obsahujícím čas.
12. Vytvořte funkci s `varargin`, která spočítá součet všech číselných vstupů.
13. Vytvořte funkci s `varargout`, která vrátí libovolný počet statistik ze vektoru (min, max, mean, median).
14. Vytvořte strukturu, jejíž pole jsou cell arrays (např. `S.days = {'Mon','Tue',...´`), a přistupte k elementu vnořené cell.
15. Najděte v poli struktur index osoby s nejvyšším průměrem teplot.
16. Načtěte CSV `teplota.csv` (sloupce: `date`, `time`, `temp`) do `table`.
17. Spojte `date` a `time` do jednoho `datetime` sloupce.
18. Seskupte záznamy podle dne a spočítejte denní průměry teplot.
19. Uložte výsledný seznam `day, meanTemp` do souboru `daily`$_{m}ean.csv.Vykresletečasovouřadudenníchprům$

2.6.2 Řešení (kompaktní MATLAB kód)

```
%% 1. Struktura person
person.name = 'Jan'; person.age = 30; person.tempHistory = [2.3
    3.1 2.8];

%% 2. Přidání pole location
person.location = 'Praha';

%% 3. Pole struktur persons
persons(1).name='Jan'; persons(1).age=30; persons(1).tempHistory
    =[2.3 3.1 2.8];
persons(2).name='Eva'; persons(2).age=25; persons(2).tempHistory
    =[1.9 2.0 2.5];
persons(3).name='Petr'; persons(3).age=40; persons(3).tempHistory
    =[3.0 2.7 2.9];

%% 4. Průměrné teploty pro každou osobu
for k=1:length(persons)
    persons(k).avgTemp = mean(persons(k).tempHistory);
end

%% 5. Cell pole C
C = {42, 'Station A', [2.3 2.5 3.1]};

%% 6. Přístup k druhému prvku cell
txt = C{2};

%% 7. cell2mat pokud jsou numerické vektory
numCell = {[1 2],[3 4]};
M = cell2mat(numCell');

%% 8. Vnořená struktura station
station.info.name = 'St1';
station.data.temps = [1.8 2.0 2.4];

%% 9. Uložení a načtení mat
save('persons.mat','persons'); clear persons; load('persons.mat')
    ;

%% 10. Export průměrů do ASCII
avg = arrayfun(@(s) s.avgTemp, persons)';
names = {persons.name}';
T = table(names, avg, 'VariableNames', {'Name','AvgTemp'});
writetable(T,'avgTemps.txt','Delimiter','\t');

%% 11. Funkce saveWithTimestamp (ukázka volání)
% saveWithTimestamp(persons);

%% 12. Funkce se varargin - součet všech vstupů
% function s = sumAll(varargin)
```

```
%    s = 0;
%    for k=1:nargin, s = s + varargin{k}; end
% end

%% 13. Funkce s varargout - statistiky
% function varargout = statsVec(v)
%    if nargout>=1, varargout{1}=min(v); end
%    if nargout>=2, varargout{2}=max(v); end
%    if nargout>=3, varargout{3}=mean(v); end
%    if nargout>=4, varargout{4}=median(v); end
% end

%% 14. Struktura s poli typu cell
S.days = {'Mon','Tue','Wed'}; S.values = { [1 2], [3 4], [5 6] };
elem = S.values{2}(1); % přístup

%% 15. Index osoby s nejvyšším průměrem
avgs = arrayfun(@(s) s.avgTemp, persons);
[~, idxMax] = max(avgs);

%% 16. Načtení CSV do table
T = readtable('teplota.csv');

%% 17. Spojení date+time na datetime
T.Datetime = datetime(strcat(string(T.date),' ',string(T.time)),
    'InputFormat','yyyy-MM-dd HH:mm:ss');

%% 18. Seskupení podle dne a výpočet denních průměrů
[G, days] = findgroups(dateshift(T.Datetime,'start','day'));
dailyMean = splitapply(@mean, T.temp, G);

%% 19. Uložení denních průměrů do CSV
writetable(table(days,dailyMean),'daily_mean.csv');

%% 20. Vykreslení časové řady a uložení PNG
figure; plot(days,dailyMean,'-o','LineWidth',1.5); xlabel('Den');
    ylabel('Průměr [ C ]');
title('Denní průměry'); grid on; legend('Průměr');
saveas(gcf,'daily_mean.png');
```

Listing 2.6: Sobota: úlohy 1–20 — kompaktní řešení

2.6.3 Řešení s podrobným vysvětlením kódu

Níže jsou jednotlivé úlohy s komentovaným kódem a podrobným vysvětlením principu.

20l. **Vytvoření struktury `person`**

```
person.name = 'Jan';
person.age = 30;
person.tempHistory = [2.3 3.1 2.8];
```

Proč: Struktura v MATLABu umožňuje pojmenované pole (fields). Každé pole může obsahovat libovolný typ (čísla, vektor, řetězec). Toto je základní způsob, jak seskupit atributy jedné entity.

2. **Přidání pole `location`**

```
person.location = 'Praha';
```

Proč: Ke struktuře lze dynamicky přidávat nová pole. To je jednoduché a čitelné pro ukládání metadat.

3. **Pole struktur `persons` pro tři osoby**

```
persons(1).name='Jan'; persons(1).age=30; persons(1).
   tempHistory=[2.3 3.1 2.8];
persons(2).name='Eva'; persons(2).age=25; persons(2).
   tempHistory=[1.9 2.0 2.5];
persons(3).name='Petr'; persons(3).age=40; persons(3).
   tempHistory=[3.0 2.7 2.9];
```

Proč: Pole struktur je typicky použitelné pro tabulková data, kde každý záznam má stejná jména polí. Umožňuje elegantní přístup pomocí `[persons.age]` nebo `arrayfun`.

4. **Výpočet průměrné teploty a uložení do struktury**

```
for k=1:length(persons)
    persons(k).avgTemp = mean(persons(k).tempHistory);
end
```

Proč: Smyčka `for` iteruje přes záznamy a doplňuje pole `avgTemp`. Alternativou je `arrayfun`, ale smyčka je jasná a přehledná.

5. **Cell pole s heterogenními daty**

```
C = {42, 'Station A', [2.3 2.5 3.1]};
```

Proč: `cell` pole ukládá heterogenní prvky — čísla, řetězce i vektory. Je užitečné, když nemáte jednotný datový typ.

6. **Přístup k druhému prvku cell**

```
txt = C{2};
```

Proč: Indexace s `{}` vrací obsah bunky (na rozdíl od `()` který vrací cell pole). Získáme přímo řetězec.

7. **Spojení číselných buněk pomocí `cell2mat`**

```
numCell = {[1 2],[3 4]};
M = cell2mat(numCell');
```

Proč: `cell2mat` převádí cell obsahující číselné bloky do jedné matice. V příkladu se použije transpozice, aby se správně spojily řádky/sloupce.

8. **Vnořená struktura `station`**

```
station.info.name = 'St1';
station.data.temps = [1.8 2.0 2.4];
```

Proč: Vnořené struktury umožňují hierarchické uložení (metadata vs. data). Je to blízké JSON/struct paradigmatům.

9. **Uložení a načtení MAT souboru**

```
save('persons.mat','persons');
clear persons;
load('persons.mat');
```

Proč: `save` serializuje proměnné do binárního MAT formátu, `load` je obnoví. `clear` ukazuje, že data skutečně přicházejí ze souboru.

10. **Export průměrů do ASCII (text)**

```
avg = arrayfun(@(s) s.avgTemp, persons)';
names = {persons.name}';
T = table(names, avg, 'VariableNames', {'Name','AvgTemp'});
writetable(T,'avgTemps.txt','Delimiter','\t');
```

Proč: `arrayfun` extrahuje pole hodnot, `table` formátuje data pro export, `writetable` ukládá do textového souboru (tab-delimited). Použitelné pro sdílení s jinými nástroji.

11. **Uložení s časovým razítkem — koncept funkce**

```
function saveWithTimestamp(x)
    fname = ['res_' datestr(now,'yyyymmdd_HHMMSS') '.mat'];
    save(fname,'x');
end
```

Proč: Automatické jméno zabraňuje přepisům a usnadňuje správu verzí výstupů. `datestr(now,...)` vytváří deterministický timestamp.

12. **Funkce s `varargin` — součet všech vstupů**

```
function s = sumAll(varargin)
    s = 0;
    for k=1:nargin
        s = s + varargin{k};
    end
end
```

Proč: `varargin` je cell pole všech dodaných argumentů; `nargin` udává jejich počet. Tato funkce ilustruje variabilní vstupy.

13. **Funkce s `varargout` — statistiky**

```
function varargout = statsVec(v)
    if nargout>=1, varargout{1}=min(v); end
    if nargout>=2, varargout{2}=max(v); end
    if nargout>=3, varargout{3}=mean(v); end
    if nargout>=4, varargout{4}=median(v); end
end
```

Proč: `varargout` umožňuje volitelný počet výstupů — uživatel si žádá tolik statistik, kolik potřebuje.

14. **Struktura s poli typu cell a přístup k vnořené cell**

```
S.days = {'Mon','Tue','Wed'};
S.values = { [1 2], [3 4], [5 6] };
elem = S.values{2}(1);
```

Proč: Kombinace struktur a cell arrays je flexibilní — umožní ukládat heterogenní kolekce v rámci záznamu.

15. **Index s nejvyšším průměrem**

```
avgs = arrayfun(@(s) s.avgTemp, persons);
[~, idxMax] = max(avgs);
```

Proč: `arrayfun` je pohodlný způsob, jak extrahovat pole z pole struktur; `max` vrací index nejlepšího.

16. **Načtení CSV do `table`**

```
T = readtable('teplota.csv');
```

Proč: `readtable` flexibilně načte CSV a mapuje sloupce do proměnných; je to vhodné pro následující datové operace.

17. **Spojení `date` a `time` do `datetime`**

```
T.Datetime = datetime(strcat(string(T.date),' ',string(T.time
    )), ...
                      'InputFormat','yyyy-MM-dd HH:mm:ss');
```

Proč: `datetime` je centrální typ pro časové analýzy; spojení řetězců je standardní postup.

18. **Seskupení podle dne a výpočet denních průměrů**

```
[G, days] = findgroups(dateshift(T.Datetime,'start','day'));
dailyMean = splitapply(@mean, T.temp, G);
```

Proč: `findgroups` + `splitapply` je efektivní idiom pro skupinové agregace v MATLABu. `dateshift(...,'start','day')` zajistí, že všechny záznamy téhož kalendářního dne spadají do jedné skupiny.

19. **Uložení denních průměrů do CSV**

```
writetable(table(days,dailyMean),'daily_mean.csv');
```

Proč: Exportujeme výsledky pro další zpracování mimo MATLAB.

20. **Vykreslení časové řady a uložení obrázku**

```
figure; plot(days,dailyMean,'-o','LineWidth',1.5);
xlabel('Den'); ylabel('Průměr [ C ]'); title('Denní průměry')
    ; grid on;
legend('Průměr');
saveas(gcf,'daily_mean.png');
```

Proč: Grafická kontrola je základní krok validace výsledků; `saveas` uloží aktuální figuru do PNG.

2.6.4 Hlavní projekt: Kompletní skript/funkce `analyzujTeplotu` (soubor `day6_project.m`)

```
function varargout = analyzujTeplotu(csvFile, varargin)
% ANALYZUJTEPLOTU  Načte CSV s daty (date,time,temp), spočítá denní průměry,
% vykreslí graf a volitelně uloží výsledky.
%
%  [minT,maxT,avgAll] = analyzujTeplotu('teplota.csv','uloz')
%
% Volitelný vstup: 'uloz'  -> uloží vysledky do daily_mean.csv a vysledky.mat
% Výstupy (varargout): 1:min, 2:max, 3:celkový průměr

    arguments
        csvFile (1,:) char
    end

    % 1) Načtení tabulky
    T = readtable(csvFile);

    % 2) Robustní převod sloupců na řetězce a spojení
    % (převod na string zajišťuje konzistenci i když jsou sloupce typu cell)
    dtStrings = strcat(string(T.date), " ", string(T.time));
    % Pokus o rozpoznání různých formátů (pokud bude potřeba, rozšířit)
    T.Datetime = datetime(dtStrings, 'InputFormat','yyyy-MM-dd HH:mm:ss', 'Locale','en_US');

    % 3) Extrakce teplot (předpokládáme sloupec 'temp')
    temp = T.temp;

    % 4) Seskupení podle kalendárních dnů
    [G, days] = findgroups(dateshift(T.Datetime,'start','day'));

    % 5) Výpočet denních průměrů (ignorujeme NaN hodnoty automaticky)
    dailyMean = splitapply(@(x) mean(x,'omitnan'), temp, G);

    % 6) Uložení výsledků do struktury a cell
    Vysledky.days = days;
    Vysledky.dailyMean = dailyMean;
    CellVysledky = {days, dailyMean};

    % 7) Vykreslení výsledku
    hFig = figure('Name','Denní průměry teplot','NumberTitle','off');
    plot(days, dailyMean, '-o', 'LineWidth', 1.6); hold on;
    grid on;
    xlabel('Den'); ylabel('Průměrná teplota [ C ]');
```

```
42      title('Denní průměry teplot');
43      legend('Denní průměr','Location','best');
44      datetick('x','yyyy-mm-dd','keepticks'); % lepší vykreslení
            osy x při potřebě
45      hold off;
46
47      % 8) Volitelně ukládáme, pokud volitelné argumenty obsahují '
            uloz'
48      if any(strcmp(varargin,'uloz'))
49          writetable(table(days,dailyMean),'daily_mean.csv');
50          save('daily_mean.mat','Vysledky','CellVysledky');
51          % také uložíme obrázek
52          saveas(hFig,'daily_mean.png');
53          disp('Výsledky uloženy: daily_mean.csv, daily_mean.mat,
                daily_mean.png');
54      end
55
56      % 9) Vrácení statistik přes varargout
57      if nargout >= 1, varargout{1} = min(dailyMean); end
58      if nargout >= 2, varargout{2} = max(dailyMean); end
59      if nargout >= 3, varargout{3} = mean(dailyMean); end
60
61  end
```

Listing 2.7: Soubor: day6_project.m¯kompletnífunkceanalyzujTeplotu

2.6.5 Podrobné vysvětlení hlavní funkce

1. `arguments` (volitelně) — zajišťuje základní typovou kontrolu argumentů (novější MATLAB). Pokud nemáte `arguments`, lze volitě odstranit první blok a použít prosté `csvFile` bez kontroly.

2. `readtable` načte CSV do `table`. Tabulka je flexibilní datová struktura pro heterogenní sloupce.

3. `datetime` — spojíme datum a čas do jednoho sloupce. `InputFormat` by měl odpovídat přesnému formátu v souboru; pokud formát není pevný, lze použít `datetime(dtStrings,'ConvertF`
nebo pokusit se o více formátů.

4. `findgroups` + `dateshift` — typický idiom pro seskupení podle kalendárního dne. `days` bude pole typu `datetime` obsahující začátky dnů.

5. `splitapply(@mean,...)` — aplikuje funkci na každou skupinu; použití anonymní funkce `@(x) mean(x,'omitnan')` ignoruje NaN hodnoty.

6. Výsledek ukládáme dvojím způsobem: do `struct` (čitelná jména polí) a do `cell` (rychlé iterace nebo export).

7. Graf: standardní `plot` s `datetick` pro srozumitelnou osu času. `saveas` uloží aktuální figuru jako PNG.

8. `varargin` kontroluje volitelný parametr `'uloz'` — pokud uživatel chce ukládat, funkcionalita se spustí.

9. `varargout` umožňuje vrátit až tři základní statistiky (min, max, celkový průměr) pro okamžité použití v okolním skriptu.

2.6.6 Poznámky a doporučení

- Pokud CSV obsahuje jiné názvy sloupců (např. `Date`, `Time`, `Temperature`), upravte přístup k sloupcům (`T.Date`, `T.Temperature`).

- Pro robustnost zvažte kontrolu chyb (existence souboru, správný formát dat), např. `if  isfile(csvFile), error('File not found'); end`.

- Pokud dataset obsahuje velké množství záznamů, `findgroups/splitapply` jsou paměťově a časově efektivní; v extrémních případech lze použít `datastore` a `tall` arrays.

- Ukládání pomocí `save` zachová datové typy; textové soubory (CSV, TXT) jsou vhodné pro sdílení mimo MATLAB.

2.7 Neděle – Recapitulace: Mini knihovna funkcí pro datové výpočty

Cílem tohoto dne je shrnout znalosti o funkcích a vytvořit vlastní malou knihovnu, která obsahuje 4 funkce pro základní práci s daty:

1. Výpočet průměru (bez použití `mean`).
2. Normalizace na interval $[0, 1]$.
3. Klouzavý průměr s nastavitelným oknem.
4. Souhrn základních statistik (min, max, průměr, medián).

Každá funkce bude mít vlastní úkol, řešení (MATLAB kód) a detailní vysvětlení. Nakonec připravíme demonstrační skript, který všechny funkce použije.

2.7.1 1) Funkce `meanValue.m`

Úkol: Vytvořte funkci, která spočítá aritmetický průměr zadaného vektoru (ručně, bez použití `mean`).

Řešení

```
function m = meanValue(x)
% meanValue - vypocet aritmetickeho prumeru
%   m = meanValue(x) vrati prumer prvku vektoru x.

n = length(x);        % pocet prvku
s = sum(x);           % soucet vsech prvku
m = s / n;            % podil souctu a poctu prvku
end
```

Řešení s podrobným vysvětlením

1. `n = length(x);` – zjistí počet prvků vektoru.
2. `s = sum(x);` – sečteme všechny prvky.
3. `m = s / n;` – vydělením součtu počtem prvků dostaneme průměr.

Proč: Aritmetický průměr $\frac{1}{n}\sum_{i=1}^{n} x_i$ je základní operace a ruční implementace ukazuje princip funkce `mean`.

2.7.2 2) Funkce `normalizeData.m`

Úkol: Vytvořte funkci, která převede vstupní vektor na hodnoty v intervalu $[0, 1]$.

Řešení

```
function y = normalizeData(x)
% normalizeData - normalizace na interval [0,1]
%   y = normalizeData(x) vrati vektor x prevedeny
%   na interval [0,1].

xmin = min(x);
xmax = max(x);
y = (x - xmin) / (xmax - xmin);
end
```

Řešení s podrobným vysvětlením

1. `xmin = min(x);` – nejmenší hodnota.
2. `xmax = max(x);` – největší hodnota.
3. `y = (x - xmin)/(xmax - xmin);` – lineární transformace do intervalu $[0, 1]$.

Proč: Normalizace umožňuje porovnávat různá data na společné škále.

2.7.3 3) Funkce `movingAverage.m`

Úkol: Vytvořte funkci, která spočítá klouzavý průměr s délkou okna k.

Řešení

```
function y = movingAverage(x, k)
% movingAverage - klouzavy prumer
%   y = movingAverage(x,k) vrati vektor y,
%   kde kazdy prvek je prumer z k sousednich prvku.

n = length(x);
y = zeros(1,n);       % predalokace
for i = 1:n
    i1 = max(1, i-k+1);     % zacatek okna
    i2 = i;                 % konec okna
    y(i) = sum(x(i1:i2)) / (i2 - i1 + 1);
end
end
```

Řešení s podrobným vysvětlením

1. Smyčka `for` prochází všechny prvky vektoru.
2. `i1 = max(1, i-k+1);` – zajistí, že okno nezačne před prvním prvkem.
3. `i2 = i;` – konec okna je aktuální pozice.
4. `sum(x(i1:i2))/(i2-i1+1);` – průměr hodnot v okně.

Proč: Klouzavý průměr vyhlazuje data a zmenšuje šum.

2.7.4 4) Funkce `dataSummary.m`

Úkol: Vytvořte funkci, která vrátí základní statistiky (minimum, maximum, průměr, medián) jako strukturu.

Řešení

```
function S = dataSummary(x)
% dataSummary - zakladni statistiky
%   S = dataSummary(x) vrati strukturu S
%   s poli min, max, mean, median.

S.min = min(x);
S.max = max(x);
S.mean = mean(x);
S.median = median(x);
end
```

Řešení s podrobným vysvětlením

1. Struktura `S` má pole `min`, `max`, `mean`, `median`.
2. Používáme vestavěné funkce `min`, `max`, `mean`, `median`.

Proč: Struktury umožňují vracet více souvisejících hodnot pohodlně v jedné proměnné.

2.7.5 Demonstrační skript `demo_library.m`

Úkol: Vyzkoušejte všechny čtyři funkce na testovacím vektoru.

Řešení

```
%% Demo skript pro mini knihovnu
x = [5 7 3 9 10 2 6];   % testovaci vektor

% 1) Vypocet prumeru
m = meanValue(x);

% 2) Normalizace
y = normalizeData(x);

% 3) Klouzavy prumer (okno 3)
ma = movingAverage(x,3);

% 4) Souhrn statistik
S = dataSummary(x);

% Vysledky
disp('Prumer:'); disp(m);
disp('Normalizovana data:'); disp(y);
disp('Klouzavy prumer:'); disp(ma);
```

```
20 disp('Souhrn:'); disp(S);
```

Výstup MATLABu

```
Prumer:
    6

Normalizovana data:
    0.4286    0.7143    0.1429    1.0000    1.1429         0    0.5714

Klouzavy prumer:
    5.0000    6.0000    5.0000    6.3333    7.3333    7.0000    6.0000

Souhrn:
    min: 2
    max: 10
    mean: 6
    median: 6
```

Řešení s podrobným vysvětlením

1. Vytvoříme testovací data `x`.
2. Funkce `meanValue` vrátí průměr.
3. `normalizeData` převede data na $[0, 1]$.
4. `movingAverage` s oknem 3 vyhladí data.
5. `dataSummary` vrátí strukturu se základními statistikami.
6. Výstupy se vypíšou v MATLABu, jak je uvedeno výše.

Proč: Tento skript ukazuje, jak může více menších funkcí tvořit mini knihovnu pro práci s daty. Každá funkce je jednoduchá, ale dohromady tvoří flexibilní nástroj.

Kapitola 3

3 Týden – Grafy a vizualizace

3.1 Pondělí – 15. den: 2D grafy (plot, scatter)

Dnes se zaměříme na vykreslování dvourozměrných grafů v MATLABu. Použijeme základní příkazy `plot` a `scatter` a ukážeme si různé možnosti, od jednoduchých grafů až po úpravu stylů, barev a více datových sérií.

3.1.1 Úkoly

1. **První 2D graf:** Vykreslete funkci $y = x$ pro $x = 0 : 10$.

   Řešení (MATLAB):

```
x = 0:10;
y = x;
plot(x,y);
```

   Vysvětlení: Použití `plot(x,y)` vykreslí body spojené čarou. Zde se jedná o diagonálu.

2. **Graf kvadratické funkce:** Vykreslete $y = x^2$.

```
x = -5:0.1:5;
y = x.^2;
plot(x,y);
```

   Vysvětlení: Operátor $.^2 provádí mocnění po prvcích.$

3. **Více funkcí v jednom grafu:** Vykreslete $y = x$ a $y = x^2$.

```
x = -5:0.1:5;
plot(x,x,'r',x,x.^2,'b');
```

   Vysvětlení: Funkce `plot` umožňuje více dvojic (x, y) a volitelné barvy (`'r'` = červená, `'b'` = modrá).

4. **Přidání názvů os a titulku:**

```
x = -5:0.1:5;
y = x.^2;
plot(x,y);
xlabel('x');
ylabel('y = x^2');
title('Parabola');
```

   Vysvětlení: `xlabel`, `ylabel`, `title` přidávají popisky.

5. **Mřížka v grafu:**

```
x = -5:0.1:5;
y = sin(x);
plot(x,y);
grid on;
```

   Vysvětlení: `grid on` zapíná mřížku.

6. **Legendy:** Přidejte legendu pro funkce $y = x$ a $y = x^2$.

```
x = -5:0.1:5;
plot(x,x,'r',x,x.^2,'b');
legend('y = x','y = x^2');
```

Vysvětlení: `legend` zobrazuje popisky datových sérií.

7. **Scatter plot základní:** Náhodné body v rovině.

```
x = rand(1,50);
y = rand(1,50);
scatter(x,y);
```

Vysvětlení: `scatter` vykreslí body bez spojení čarami.

8. **Scatter s barvou a velikostí bodů:**

```
x = rand(1,50);
y = rand(1,50);
s = 100*rand(1,50);      % velikosti bodů
c = rand(1,50);          % barvy bodů
scatter(x,y,s,c,'filled');
```

Vysvětlení: Parametry `s` a `c` ovlivňují vzhled bodů.

9. **Více scatter vrstev v jednom grafu:**

```
x1 = randn(1,50);
y1 = randn(1,50);
x2 = randn(1,50)+3;
y2 = randn(1,50)+3;
scatter(x1,y1,'r');
hold on;
scatter(x2,y2,'b');
hold off;
```

Vysvětlení: `hold on` umožňuje kreslit více sad bodů do stejného obrázku.

10. **Spojení plot a scatter:**

```
x = 0:0.1:10;
y = sin(x);
plot(x,y,'b');
hold on;
scatter(x,y,'r','filled');
hold off;
```

Vysvětlení: Čára a body v jednom grafu.

11. **Nelineární funkce:** $y = e^{-x}\cos(x)$.

```
x = 0:0.1:10;
y = exp(-x).*cos(x);
plot(x,y,'m');
```

Vysvětlení: Exponenciálně tlumené oscilace.

12. **Více subgrafů:** Dva grafy vedle sebe.

```
x = 0:0.1:10;
subplot(1,2,1);
plot(x,sin(x));
subplot(1,2,2);
plot(x,cos(x));
```

Vysvětlení: `subplot(m,n,p)` vytvoří mřížku grafů.

13. **Scatter s průhledností:**

```
x = randn(1,200);
y = randn(1,200);
scatter(x,y,50,'b','filled','MarkerFaceAlpha',0.3);
```

Vysvětlení: Parametr `MarkerFaceAlpha` nastavuje průhlednost.

14. **Kombinace sinusů:** $y = \sin(x) + 0.5\sin(3x)$.

```
x = 0:0.01:10;
y = sin(x)+0.5*sin(3*x);
plot(x,y,'k');
```

Vysvětlení: Superpozice dvou harmonických funkcí.

15. **Graf s omezenými osami:**

```
x = -10:0.1:10;
y = x.^3;
plot(x,y);
xlim([-5 5]);
ylim([-50 50]);
```

Vysvětlení: `xlim`, `ylim` omezují rozsah zobrazení.

16. **Více stylů čar:**

```
x = 0:0.1:10;
plot(x,sin(x),'--r',x,cos(x),':b',x,tan(x),'-g');
ylim([-5 5]);
```

Vysvětlení: Styly čar `'-'`, `':'` , `'-'` a barvy.

17. **Scatter s barevnou mapou:**

```
x = rand(1,100);
y = rand(1,100);
c = y;  % barva podle y
scatter(x,y,50,c,'filled');
colorbar;
```

Vysvětlení: `colorbar` přidává legendu barev.

18. **Srovnání rychlosti plot vs. scatter:**

```
x = rand(1,1e5);
y = rand(1,1e5);
subplot(1,2,1); plot(x,y,'.');
subplot(1,2,2); scatter(x,y,'.');
```

Vysvětlení: `plot(...,'.')` a `scatter` se chovají podobně, ale mají různé rychlosti a možnosti.

19. **Spojení více grafů s legendou:**

```
x = 0:0.1:10;
plot(x,sin(x),'r',x,cos(x),'g',x,sin(x).*cos(x),'b');
legend('sin(x)','cos(x)','sin(x)cos(x)');
```

Vysvětlení: Více funkcí se sdílenou legendou.

20. **Scatter simulace šumu:**

```
x = 0:0.1:10;
y = sin(x) + 0.2*randn(size(x));
plot(x,sin(x),'b'); hold on;
scatter(x,y,'r');
hold off;
```

Vysvětlení: Červené body simulují měřená data se šumem, modrá křivka je ideální průběh.

3.1.2 Shrnutí: Plot vs. Scatter

Funkce	Popis	Typické použití
`plot(x,y)`	Spojí body čarou	Funkční závislosti, spojité průběhy
`scatter(x,y)`	Vykreslí izolované body	Experimentální data, shluky bodů
`plot(...,'.')`	Body místo čáry	Rychlejší než `scatter`, méně možností
`scatter(...,s,c)`	Body s velikostí a barvou	Vizuální zvýraznění 3. a 4. proměnné

3.2 Úterý – 16. den: Nastavení os, popisky, legendy, 2D + 3D grafy

Dnes se zaměříme na možnosti přizpůsobení grafů: nastavení os, popisky, legendy. Ukážeme si také základní 3D grafy v MATLABu: `mesh`, `surf`, `contour`.

3.2.1 Úkoly

1. **Nastavení rozsahu os v 2D:**

```
x = -10:0.1:10;
y = sin(x);
plot(x,y);
xlim([-5 5]);
ylim([-1 1]);
```

Vysvětlení: Funkce `xlim`, `ylim` nastaví rozsah zobrazených dat.

2. **Popisky os a titulek:**

```
x = -2*pi:0.01:2*pi;
y = cos(x);
plot(x,y);
xlabel('x [rad]');
ylabel('cos(x)');
title('Cosinusová funkce');
```

Vysvětlení: Popisky zlepšují čitelnost grafu.

3. **Legenda pro více funkcí:**

```
x = 0:0.1:10;
plot(x,sin(x),'r',x,cos(x),'b');
legend('sin(x)','cos(x)');
```

Vysvětlení: Legenda identifikuje jednotlivé křivky.

4. **Logaritmická osa:**

```
x = logspace(0,2,100);
y = x.^2;
loglog(x,y);
```

Vysvětlení: `loglog` nastaví obě osy logaritmicky.

5. **Polární graf:**

```
theta = 0:0.01:2*pi;
rho = 1 + cos(theta);
polarplot(theta,rho);
```

Vysvětlení: `polarplot` zobrazí data v polárních souřadnicích.

6. **Meshgrid pro 3D výpočty:**

```
[x,y] = meshgrid(-3:0.1:3,-3:0.1:3);
z = x.^2 + y.^2;
mesh(x,y,z);
```

Vysvětlení: `meshgrid` vytváří matici souřadnic pro výpočet $z = f(x, y)$.

7. **Základní 3D graf pomocí mesh:**

```
[x,y] = meshgrid(-5:0.25:5,-5:0.25:5);
z = sin(sqrt(x.^2+y.^2));
mesh(x,y,z);
```

Vysvětlení: `mesh` vykresluje 3D síť.

8. **Graf pomocí surf:**

```
[x,y] = meshgrid(-5:0.25:5,-5:0.25:5);
z = sin(x).*cos(y);
surf(x,y,z);
```

Vysvětlení: `surf` vykresluje hladký povrch vyplněný barvami.

9. **Barevná mapa povrchu:**

```
[x,y] = meshgrid(-3:0.1:3,-3:0.1:3);
z = x.*exp(-x.^2 - y.^2);
surf(x,y,z);
colormap jet;
colorbar;
```

Vysvětlení: Barevná mapa vizualizuje hodnoty z.

10. **3D konturový graf:**

```
[x,y] = meshgrid(-3:0.1:3,-3:0.1:3);
z = peaks(x,y);
contour(x,y,z,20);
```

Vysvětlení: `contour` vykresluje vrstevnice.

11. **Contourf – vyplněné vrstevnice:**

```
[x,y] = meshgrid(-3:0.05:3,-3:0.05:3);
z = x.^2 - y.^2;
contourf(x,y,z,30);
colorbar;
```

Vysvětlení: `contourf` vykreslí vyplněné oblasti mezi vrstevnicemi.

12. **Kombinace surf + contour:**

```
[x,y] = meshgrid(-3:0.1:3,-3:0.1:3);
z = sin(x).*cos(y);
surf(x,y,z);
```

```
hold on;
contour3(x,y,z,20,'k');
hold off;
```

Vysvětlení: `contour3` přidává 3D vrstevnice na povrch.

13. **Pohled a rotace grafu:**

```
[x,y] = meshgrid(-5:0.25:5,-5:0.25:5);
z = x.^2 - y.^2;
surf(x,y,z);
view(45,30);
```

Vysvětlení: Funkce `view(az,el)` nastavuje azimut a elevaci.

14. **Zobrazení mřížky os:**

```
sphere;
grid on;
```

Vysvětlení: `grid on` doplní čtvercovou mřížku.

15. **Shading (stínování):**

```
[x,y] = meshgrid(-3:0.1:3,-3:0.1:3);
z = sin(x).*cos(y);
surf(x,y,z);
shading interp;
```

Vysvětlení: `shading interp` vyhladí přechody barev.

16. **Osy v logaritmické stupnici:**

```
x = logspace(0,2,100);
y = sqrt(x);
semilogx(x,y);
```

Vysvětlení: Použití logaritmické osy pouze pro osu x.

17. **Více 3D grafů vedle sebe:**

```
[x,y] = meshgrid(-3:0.1:3);
z1 = sin(x).*cos(y);
z2 = peaks(x,y);

subplot(1,2,1); mesh(x,y,z1);
subplot(1,2,2); surf(x,y,z2);
```

Vysvětlení: Subplot umožňuje srovnávat různé povrchy.

18. **Použití funkcí axis:**

```
[x,y] = meshgrid(-2:0.1:2);
z = x.^2 + y.^2;
surf(x,y,z);
axis equal;
```

Vysvětlení: `axis equal` zajistí stejnou měřítkovou jednotku na osách.

19. **Mesh vs. Surf s colormap:**

```
[x,y] = meshgrid(-3:0.1:3,-3:0.1:3);
z = sin(sqrt(x.^2+y.^2));
subplot(1,2,1); mesh(x,y,z);
subplot(1,2,2); surf(x,y,z); colormap jet;
```

Vysvětlení: Rozdíl mezi drátěným a vyplněným povrchem.

20. **3D scatter:**

```
x = randn(1,100);
y = randn(1,100);
z = x.^2 + y.^2;
scatter3(x,y,z,50,z,'filled');
colorbar;
```

Vysvětlení: `scatter3` vykresluje body v prostoru, barva odpovídá z.

3.2.2 Shrnutí

Funkce	Popis	Použití
`plot`	2D čárový graf	spojité funkce
`scatter`	2D body	experimentální data
`mesh`	3D drátěný povrch	rychlá vizualizace
`surf`	3D barevný povrch	detailní vizualizace
`contour`	2D vrstevnice	mapy, úrovně funkcí
`contour3`	3D vrstevnice	kombinace s povrchem
`scatter3`	3D body	prostorová data

3.3 Středa – 17. den: 3D grafy, rozsahy a mřížky, popisky a anotace

Tento den se soustředíme na 3D grafy v MATLABu: `plot3`, `mesh`, `surf`, `contour`, `contour3`, `slice`, `isosurface`. Také si procvičíme práci s osami (`axis`, `xlim`, `ylim`, `zlim`), mřížkou (`grid on/off`, `hold on/off`) a popisky (`xlabel`, `ylabel`, `zlabel`, `title`, `legend`, `text`, `annotation`).

3.3.1 Úkoly

1. **Základní 3D čára pomocí plot3:**

```
t = 0:0.1:10;
x = cos(t); y = sin(t); z = t;
plot3(x,y,z);
```

   Proč: `plot3` vykresluje 3D trajektorii, zde šroubovice.

2. **Zobrazení více 3D křivek:**

```
t = 0:0.1:10;
plot3(cos(t),sin(t),t,'r',cos(t),-sin(t),t,'b');
legend('Spirála 1','Spirála 2');
```

   Proč: Legenda pomáhá rozlišit více křivek.

3. **3D mřížka pomocí meshgrid:**

```
[x,y] = meshgrid(-5:0.5:5);
z = sin(sqrt(x.^2+y.^2));
mesh(x,y,z);
```

   Proč: `mesh` zobrazí povrch jako drátěný model.

4. **Vyplněný 3D povrch – surf:**

```
[x,y] = meshgrid(-3:0.1:3);
z = x.*exp(-x.^2 - y.^2);
surf(x,y,z);
```

   Proč: `surf` vykreslí hladký barevný povrch.

5. **Vrstevnicový graf – contour:**

```
[x,y] = meshgrid(-3:0.1:3);
z = x.^2 - y.^2;
contour(x,y,z,20);
```

   Proč: Vrstevnice zobrazují úrovně funkce $z = f(x, y)$.

6. **3D vrstevnice – contour3:**

```
[x,y] = meshgrid(-3:0.1:3);
z = peaks(x,y);
contour3(x,y,z,30);
```

Proč: `contour3` přidá výšku vrstevnic.

7. **Kombinace surf + contour:**

```
[x,y] = meshgrid(-3:0.1:3);
z = sin(x).*cos(y);
surf(x,y,z); hold on;
contour3(x,y,z,15,'k');
hold off;
```

Proč: Kombinace vizualizuje jak povrch, tak vrstevnice.

8. **Řezová vizualizace – slice:**

```
[x,y,z] = meshgrid(-2:0.1:2);
v = x.^2 + y.^2 + z.^2;
slice(x,y,z,v,[0],[0],[0]);
```

Proč: `slice` vykreslí řezy 3D datem v rovinách.

9. **Isosurface – plocha konstantní hodnoty:**

```
[x,y,z] = meshgrid(-2:0.1:2);
v = x.^2 + y.^2 + z.^2;
p = patch(isosurface(x,y,z,v,2));
isonormals(x,y,z,v,p);
set(p,'FaceColor','red','EdgeColor','none');
camlight; lighting gouraud;
```

Proč: `isosurface` vykresluje 3D plochu s konstantní hodnotou pole.

10. **Omezení os pomocí xlim, ylim, zlim:**

```
[x,y] = meshgrid(-5:0.1:5);
z = sin(sqrt(x.^2+y.^2));
surf(x,y,z);
xlim([-2 2]); ylim([-2 2]); zlim([-1 1]);
```

Proč: Umožňuje zaměřit se na část dat.

11. **Přepínání mřížky:**

```
peaks; grid off;
```

Proč: `grid on/off` zapíná/vypíná mřížku os.

12. **Použití hold on/off:**

```
t = 0:0.1:10;
plot3(sin(t),cos(t),t,'r'); hold on;
plot3(sin(t),-cos(t),t,'b');
hold off;
```

Proč: `hold on` umožňuje kombinovat více grafů v jednom okně.

13. **Popisky os a titulek v 3D:**

```
[x,y] = meshgrid(-3:0.1:3);
z = x.^2 - y.^2;
surf(x,y,z);
xlabel('x'); ylabel('y'); zlabel('z');
title('Hyperbolický povrch');
```

Proč: Zlepšuje interpretaci grafu.

14. **Legenda v 3D grafu:**

```
t = 0:0.1:10;
plot3(cos(t),sin(t),t,'r',sin(t),cos(t),t,'b');
legend('Trajektorie 1','Trajektorie 2');
```

Proč: Označí více trajektorií.

15. **Text v grafu:**

```
t = 0:0.1:10;
x = cos(t); y = sin(t); z = t;
plot3(x,y,z);
text(0,0,5,'Střed spirály','Color','r');
```

Proč: `text` přidá popisek přímo do grafu.

16. **Anotace šipkou:**

```
peaks;
annotation('textarrow',[0.3 0.4],[0.7 0.8],'String','Maximum'
   );
```

Proč: `annotation` zvýrazňuje důležitý bod.

17. **Axis equal a square:**

```
sphere;
axis equal;
```

Proč: `axis equal` zajistí stejnou jednotku na všech osách.

18. **Více 3D grafů – subplot:**

```
[x,y] = meshgrid(-3:0.1:3);
z1 = sin(x).*cos(y); z2 = x.^2+y.^2;

subplot(1,2,1); mesh(x,y,z1);
subplot(1,2,2); surf(x,y,z2);
```

Proč: Umožní porovnání dvou povrchů.

19. **Kombinace plot3 + surf:**

```
[x,y] = meshgrid(-3:0.1:3);
z = sin(x).*cos(y);
surf(x,y,z); hold on;
plot3(0,0,0,'ro','MarkerSize',10,'MarkerFaceColor','r');
hold off;
```

Proč: Zvýrazní specifický bod na povrchu.

3.3.2 Shrnutí

Funkce	Popis	Použití
`plot3`	3D křivka	trajektorie, spirály
`mesh`	Drátěný povrch	rychlá vizualizace
`surf`	Barevný povrch	hladké plochy
`contour`, `contour3`	Vrstevnice	úrovně funkce
`slice`	Řezy objemem	3D data
`isosurface`	Plocha konstantní hodnoty	3D vizualizace polí
`axis, xlim, ylim, zlim`	Osy a rozsahy	přizpůsobení pohledu
`xlabel, ylabel, zlabel, title`	Popisky	popisy os
`legend, text, annotation`	Anotace	vysvětlivky, šipky

3.4 Čtvrtek – 18. den: Více grafů v jednom okně, profesionální prezentace výsledků

Dnes se zaměříme na tvorbu vícenásobných grafů (`subplot`, `tiledlayout`) a na profesionální formátování výstupů pro prezentace a vědecké články. Ukážeme si použití LaTeXu v popiscích, konzistentní stylování (tloušťky čar, barvy, fonty) a export grafů do různých formátů (PNG, PDF, EPS).

3.4.1 Úkoly

1. **Základní použití subplot:**

```
x = 0:0.1:10;
subplot(2,1,1);
plot(x,sin(x));
subplot(2,1,2);
plot(x,cos(x));
```

   Proč: `subplot(m,n,p)` rozdělí okno na m řádků, n sloupců a vykreslí graf do panelu p.

2. **Více grafů v řadě:**

```
x = 0:0.1:10;
subplot(1,3,1); plot(x,sin(x));
subplot(1,3,2); plot(x,cos(x));
subplot(1,3,3); plot(x,exp(-0.1*x).*sin(x));
```

   Proč: Užitečné pro porovnání více funkcí.

3. **Použití tiledlayout:**

```
x = linspace(0,10,100);
tiledlayout(2,2);
nexttile; plot(x,sin(x));
nexttile; plot(x,cos(x));
nexttile; plot(x,sin(2*x));
nexttile; plot(x,cos(2*x));
```

   Proč: `tiledlayout` je flexibilnější a modernější než `subplot`.

4. **Sdílené popisky os:**

```
x = 0:0.1:10;
t = tiledlayout(2,1);
nexttile; plot(x,sin(x));
nexttile; plot(x,cos(x));
xlabel(t,'Čas t [s]');
ylabel(t,'Hodnota');
```

   Proč: Lze dát jeden společný popisek pro všechny panely.

5. **LaTeX v popiscích:**

```
x = -2:0.1:2;
y = x.^2;
plot(x,y);
xlabel('$x$','Interpreter','latex');
ylabel('$y=x^2$','Interpreter','latex');
title('Parabola $y=x^2$','Interpreter','latex');
```

Proč: Profesionální matematická notace.

6. **Konzistentní tloušťka čar:**

```
x = 0:0.1:10;
plot(x,sin(x),'LineWidth',2);
```

Proč: `LineWidth` zlepšuje čitelnost.

7. **Konzistentní barvy (IEEE styl):**

```
x = 0:0.1:10;
plot(x,sin(x),'Color',[0 0 0]); % černá
hold on;
plot(x,cos(x),'--','Color',[0.5 0.5 0.5]); % šedá
hold off;
```

Proč: Profesionální grafy často používají omezenou paletu.

8. **Konzistentní fonty:**

```
set(gca,'FontName','Times New Roman','FontSize',14);
```

Proč: Užitečné pro jednotný styl publikací.

9. **Formátování mřížky:**

```
plot(x,sin(x));
grid on; grid minor;
set(gca,'LineWidth',1,'GridAlpha',0.3);
```

Proč: Jemná mřížka zlepšuje přehlednost.

10. **Použití tiledlayout pro asymetrické dělení:**

```
t = tiledlayout(2,2);
nexttile([2 1]); plot(x,sin(x)); % větší plocha
nexttile; plot(x,cos(x));
nexttile; plot(x,exp(-0.1*x).*sin(x));
```

Proč: Lze kombinovat panely různé velikosti.

11. **Export grafu do PNG:**

```
f = figure;
plot(x,sin(x));
saveas(f,'graf1.png');
```

Proč: PNG se hodí pro prezentace a web.

12. **Export grafu do PDF:**

```
f = figure;
plot(x,cos(x));
print(f,'graf2','-dpdf');
```

Proč: PDF je vektorový formát vhodný do LaTeXu.

13. **Export grafu do EPS:**

```
f = figure;
plot(x,exp(-0.1*x).*sin(x));
print(f,'graf3','-depsc');
```

Proč: EPS je tradiční vektorový formát pro časopisy.

14. **Export s nastavením DPI:**

```
f = figure;
plot(x,sin(2*x));
exportgraphics(f,'graf4.png','Resolution',300);
```

Proč: DPI ovlivňuje kvalitu tisku.

15. **Nastavení velikosti obrázku:**

```
f = figure('Units','inches','Position',[1 1 6 4]);
plot(x,cos(2*x));
exportgraphics(f,'graf5.pdf');
```

Proč: Užitečné pro přesné nastavení do článků.

16. **Použití LaTeX legendy:**

```
plot(x,sin(x),'r',x,cos(x),'b');
legend({'$\sin(x)$','$\cos(x)$'},'Interpreter','latex');
```

Proč: Legenda s matematickými symboly vypadá profesionálně.

17. **Více figure oken:**

```
figure(1); plot(x,sin(x));
figure(2); plot(x,cos(x));
```

Proč: Oddělení grafů do samostatných oken.

18. **Uložení a načtení figure:**

```
f = figure;
plot(x,sin(x));
savefig(f,'graf6.fig'); % uložit
openfig('graf6.fig');   % znovu otevřít
```

Proč: Uložení figure do MATLAB formátu umožňuje pozdější úpravy.

19. **Kombinace subplot + export:**

```
x = linspace(0,10,100);
subplot(2,1,1); plot(x,sin(x));
subplot(2,1,2); plot(x,cos(x));
print('graf7','-dpng','-r300');
```

Proč: Lze exportovat i celé okno se subplota.

20. **Profesionální příprava grafu pro článek:**

```
f = figure('Units','centimeters','Position',[5 5 12 8]);
plot(x,sin(x),'k','LineWidth',1.5);
xlabel('$x$','Interpreter','latex');
ylabel('$\sin(x)$','Interpreter','latex');
title('Ukázkový graf','Interpreter','latex');
grid on;
set(gca,'FontName','Times New Roman','FontSize',12);
exportgraphics(f,'final_plot.pdf','ContentType','vector');
```

Proč: Kombinuje LaTeX, konzistentní fonty, tloušťku čar a export do vektorového formátu.

3.4.2 Shrnutí

Funkce	Použití
`subplot`	Rozdělení okna na panely
`tiledlayout, nexttile`	Moderní správa více grafů
`saveas, print, exportgraphics`	Export do PNG, PDF, EPS
`Interpreter='latex'`	Profesionální matematické popisky
`LineWidth, FontName, FontSize`	Konzistentní styl

3.5 Pátek – 19. den: Animace v MATLABu

Dnes se budeme zabývat tvorbou animací v MATLABu. Začneme jednoduchými interaktivními postupy pomocí `pause`, přejdeme k animacím s `comet`, následně k manuálním animacím pomocí `getframe` a nakonec si ukážeme export videa pomocí `VideoWriter`.

3.5.1 Úkoly

1. **Jednoduchý pohyb bodu po kružnici s pause:**

```
theta = linspace(0,2*pi,100);
for k=1:length(theta)
    plot(cos(theta(k)),sin(theta(k)),'ro');
    axis([-1.5 1.5 -1.5 1.5]);
    pause(0.05);
end
```

   Proč: `pause` umožňuje jednoduchou animaci krok po kroku.

2. **Pohyb sinusové vlny (rolování):**

```
x = linspace(0,4*pi,200);
y = sin(x);
for k=1:50:length(x)
    plot(x(1:k),y(1:k),'b');
    axis([0 4*pi -1 1]);
    axis equal;
    pause(0.1);
end
```

   Proč: Simulace postupného vykreslování vlny.

3. **Použití comet animace:**

```
x = linspace(0,4*pi,200);
y = sin(x);
comet(x,y);
```

   Proč: `comet` automaticky kreslí stopu bodu.

4. **Comet pro 3D trajektorii:**

```
t = linspace(0,10,500);
x = cos(t); y = sin(t); z = t/10;
comet3(x,y,z);
```

   Proč: `comet3` vizualizuje pohyb v prostoru.

5. **Simulace harmonického oscilátoru:**

```
t = linspace(0,10,200);
y = cos(t);
for k=1:length(t)
    plot(t(1:k),y(1:k),'r');
```

```
    axis([0 10 -1.5 1.5]);
    pause(0.05);
end
```

Proč: Ukázka dynamiky časové řady.

6. **Rostoucí parabola:**

```
x = -2:0.1:2;
for a=1:10
    y = a*x.^2;
    plot(x,y);
    axis([-2 2 0 40]);
    pause(0.3);
end
```

Proč: Dynamická změna parametru funkce.

7. **Více sinusů s fázovým posunem:**

```
x = linspace(0,2*pi,200);
for phi=0:0.2:2*pi
    y = sin(x+phi);
    plot(x,y);
    ylim([-1 1]);
    pause(0.05);
end
```

Proč: Demonstruje posun fáze.

8. **Animace křivky Lissajous:**

```
t = linspace(0,10,500);
x = sin(3*t);
y = cos(2*t);
for k=1:length(t)
    plot(x(1:k),y(1:k),'b');
    axis([-1 1 -1 1]);
    pause(0.01);
end
```

Proč: Lissajousovy obrazce ukazují složené harmonické pohyby.

9. **Použití getframe pro snímkování:**

```
x = linspace(0,2*pi,100);
y = sin(x);
f = figure;
for k=1:length(x)
    plot(x(1:k),y(1:k),'r');
    axis([0 2*pi -1 1]);
    M(k) = getframe(f);
end
```

Proč: `getframe` ukládá jednotlivé snímky animace.

10. **Přehrání animace pomocí movie:**

```
movie(M,2,10);
```

Proč: Funkce `movie` přehraje uložené snímky, zde 2x rychlostí 10 fps.

11. **Uložení animace do AVI souboru:**

```
v = VideoWriter('sinus.avi');
open(v);
writeVideo(v,M);
close(v);
```

Proč: Export animace do video souboru.

12. **Animace 2D částice s náhodným pohybem:**

```
x = 0; y = 0;
for k=1:100
    x = x + randn*0.1;
    y = y + randn*0.1;
    plot(x,y,'ro'); hold on;
    axis([-5 5 -5 5]);
    pause(0.1);
end
```

Proč: Simulace náhodné procházky.

13. **Rotace sinusové vlny ve 3D:**

```
[x,y] = meshgrid(0:0.2:10,0:0.2:10);
z = sin(x)+cos(y);
f = figure;
for ang=0:5:360
    surf(x,y,z);
    view(ang,30);
    pause(0.1);
end
```

Proč: Pohled na 3D graf z různých úhlů.

14. **Animace amplitudy sinusové vlny:**

```
x = 0:0.1:10;
for A=0:0.1:2
    y = A*sin(x);
    plot(x,y);
    ylim([-2 2]);
    pause(0.05);
end
```

Proč: Ukazuje změnu amplitudy.

15. **Animace rozptylu v gaussovské funkci:**

```
x = -5:0.1:5;
for sigma=0.5:0.2:2
    y = exp(-x.^2/(2*sigma^2));
    plot(x,y);
    ylim([0 1]);
    pause(0.1);
end
```

Proč: Demonstruje vliv rozptylu na šířku křivky.

16. **Animace růstu spirály:**

```
theta = linspace(0,4*pi,500);
r = linspace(0,5,500);
x = r.*cos(theta);
y = r.*sin(theta);
for k=1:length(theta)
    plot(x(1:k),y(1:k));
    axis equal;
    pause(0.01);
end
```

Proč: Dynamická spirála ilustruje polární souřadnice.

17. **Dynamická změna barvy:**

```
x = linspace(0,2*pi,100);
for k=1:100
    plot(x,sin(x),'Color',[k/100 0 1-k/100],'LineWidth',2);
    ylim([-1 1]);
    pause(0.05);
end
```

Proč: Animace nejen pohybu, ale i barevného přechodu.

18. **Použití drawnow místo pause:**

```
x = 0:0.1:10;
y = sin(x);
for k=1:length(x)
    plot(x(1:k),y(1:k),'b');
    axis([0 10 -1 1]);
    drawnow;
end
```

Proč: `drawnow` okamžitě aktualizuje graf bez pauzy.

19. **Finální projekt – export spirály do MP4:**

```
theta = linspace(0,4*pi,200);
r = linspace(0,5,200);
x = r.*cos(theta); y = r.*sin(theta);
```

```
f = figure;
v = VideoWriter('spirala.mp4','MPEG-4');
open(v);
for k=1:length(theta)
    plot(x(1:k),y(1:k),'r','LineWidth',2);
    axis equal; axis([-6 6 -6 6]);
    frame = getframe(f);
    writeVideo(v,frame);
end
close(v);
```

Proč: Kompletní animace uložená do MP4 videa.

3.5.2 Shrnutí

Funkce	**Použití**
`pause`, `drawnow`	Interaktivní zpomalení / aktualizace
`comet`, `comet3`	Automatická animace trajektorie
`getframe`, `movie`	Snímkování a přehrávání animací
`VideoWriter`	Export animace do video souboru (AVI, MP4)

[a4paper,12pt]article [czech]babel [utf8]inputenc [T1]fontenc amsmath, amssymb geometry margin=2.5cm listings xcolor

3.6 20. den – Interaktivní grafy, GUI a animace

Cílem tohoto dne je seznámit se s možnostmi **interaktivity** v MATLABu:

- získávání vstupů od uživatele přímo z grafu pomocí `ginput`,
- práce s jednoduchými grafickými prvky GUI (`uicontrol`),
- tvorba malých aplikací s posuvníkem (`slider`),
- a také vytvoření **animace pohybu částice v gravitačním poli**.

Na rozdíl od předchozích dnů se nebudeme věnovat 20 malým úlohám, ale 4 komplexnějším blokům.

3.6.1 Úloha 1 – Interaktivní grafy s ginput

```
% Úloha 1: Výběr bodů v grafu pomocí ginput
x = linspace(0, 2*pi, 100);
y = sin(x);

figure;
plot(x, y, 'b-', 'LineWidth', 2);
title('Klikněte do grafu a vyberte 3 body');
xlabel('x'); ylabel('sin(x)');
grid on;

% ginput(n) umožní uživateli vybrat n bodů kliknutím
[x_click, y_click] = ginput(3);

hold on;
plot(x_click, y_click, 'ro', 'MarkerSize', 10, 'MarkerFaceColor',
    'r');
```

Vysvětlení:

- `linspace(0,2*pi,100)` vytvoří 100 rovnoměrně rozložených hodnot.
- Funkce `plot` vykreslí sinusoidu.
- `ginput(3)` čeká na to, až uživatel klikne do grafu myší – zaznamená 3 body (souřadnice `x_click`, `y_click`).
- Červené body `'ro'` zobrazí vybrané souřadnice.

Proč: Tímto způsobem lze interaktivně sbírat data nebo vybírat parametry z grafu.

3.6.2 Úloha 2 – GUI prvky (uicontrol)

```
% Úloha 2: Základní tlačítko a slider
figure;
uicontrol('Style','text', 'String','Posuň slider:',...
          'Position',[20 150 100 30]);

slider = uicontrol('Style','slider','Min',0,'Max',10,'Value',5,...
                   'Position',[150 150 200 30]);

button = uicontrol('Style','pushbutton','String','Zobraz hodnotu',...
                   'Position',[150 100 200 30],...
                   'Callback',@(src,event) disp(['Hodnota slideru: ', num2str(slider.Value)]));
```

Vysvětlení:

- `uicontrol` vytváří grafické prvky v okně.
- Typ `'text'` zobrazuje popisek.
- Typ `'slider'` vytváří posuvník s hodnotami od 0 do 10.
- Typ `'pushbutton'` definuje tlačítko. Parametr `Callback` je funkce, která se spustí po kliknutí (v tomto případě vypíše aktuální hodnotu slideru do konzole).

Proč: Základní stavební kameny pro interaktivní GUI aplikace.

3.6.3 Úloha 3 – Jednoduchý GUI skript se sliderem

```
% Úloha 3: Slider mění amplitudu sinusoidy
f = figure;
ax = axes('Parent',f);

x = linspace(0, 2*pi, 200);
amp = 1;
hPlot = plot(ax, x, amp*sin(x),'b','LineWidth',2);
xlabel('x'); ylabel('y'); grid on;
title('Amplituda se mění pomocí slideru');

slider = uicontrol('Style','slider','Min',0,'Max',5,'Value',1,...
                   'Position',[100 20 300 20]);

% Callback funkce pro slider
slider.Callback = @(src,event) set(hPlot,'YData', src.Value*sin(x));
```

Vysvětlení:

- Nejprve vykreslíme sinusoidu s amplitudou 1.

- Slider nastavuje hodnotu mezi 0 a 5.
- V `Callback` funkci se při změně slideru dynamicky mění `YData` sinusoidy – tím se interaktivně mění její amplituda.

Proč: Tohle je jednoduchý příklad propojení grafu a ovládacího prvku – uživatel vidí okamžitý efekt.

3.6.4 Úloha 4 – Animace částice v gravitačním poli

```
% Úloha 4: Animace volného pádu
g = 9.81;          % gravitační zrychlení
y0 = 50;           % počáteční výška
v0 = 0;            % počáteční rychlost
t = linspace(0, 4, 100);

y = y0 + v0*t - 0.5*g*t.^2;
y(y<0) = 0; % částice nesmí jít pod zem

figure;
h = plot(0,y0,'ro','MarkerSize',10,'MarkerFaceColor','r');
axis([0 1 0 y0+10]);
xlabel('x'); ylabel('y'); grid on;
title('Pohyb částice v gravitačním poli');

for k = 1:length(t)
    set(h,'YData',y(k));
    drawnow;
    pause(0.05);
end
```

Vysvětlení:

- Fyzikální model: $y(t) = y_0 + v_0 t - \frac{1}{2} g t^2$.
- Po dosažení země nastavíme hodnoty na 0, aby částice „zůstala“ na zemi.
- Animace využívá cyklus `for`, funkci `set` pro aktualizaci polohy a `drawnow` pro vykreslení.

Proč: Tento postup umožňuje vytvářet jednoduché animace fyzikálních jevů.

3.6.5 Shrnutí dne

- `ginput` – interaktivní výběr bodů v grafu.
- `uicontrol` – základní GUI prvky (text, tlačítko, slider).
- `Callback` – definice akcí při interakci.
- `drawnow`, `set` – nástroje pro animaci.

MATLAB tak poskytuje jednoduché možnosti, jak vytvářet nejen vědecké vizualizace, ale i interaktivní aplikace a simulace.

3.7 Neděle – Recapitulace: Mini-prezentace (PDF), Projektový blok (GUI+animace+data), 20 úloh z matematiky a fyziky

3.7.1 Část A – Mini-prezentace z grafů (export do PDF/PNG)

Cíl: vytvořit 3–4 reprezentativní grafy (2D i 3D) a korektně je *exportovat* v publikační kvalitě.

A.1 Sada grafů + export

```
% A.1: Připravíme 3 obrázky a uložíme je jako PNG i PDF
% Proč: PNG pro prezentaci/slide, PDF pro LaTeX/vektory.

% 1) 2D křivka + popisky + legenda
x = linspace(0, 2*pi, 400);
y1 = sin(x);
y2 = 0.5*cos(2*x);
f1 = figure('Color','w');            % bílé pozadí kvůli tisku
plot(x, y1, 'LineWidth', 2); hold on
plot(x, y2, '--', 'LineWidth', 2);
grid on; xlabel('$x$','Interpreter','latex');
ylabel('$y$','Interpreter','latex');
title('Srovnání $\sin(x)$ a $\tfrac{1}{2}\cos(2x)$','Interpreter','latex');
legend({'$\sin(x)$','$\tfrac{1}{2}\cos(2x)$'}, 'Interpreter','latex','Location','best');
exportgraphics(f1, 'fig_sin_cos.png', 'Resolution', 300);  % Proč: pro prezentace
exportgraphics(f1, 'fig_sin_cos.pdf');                     % Proč: vektor pro LaTeX

% 2) 3D povrch z = sin(x)*cos(y)
[xg, yg] = meshgrid(linspace(-3,3,150));
zg = sin(xg).*cos(yg);
f2 = figure('Color','w');
surf(xg, yg, zg); shading interp; colormap parula; colorbar
xlabel('$x$','Interpreter','latex'); ylabel('$y$','Interpreter','latex');
zlabel('$z$','Interpreter','latex');
title('$z=\sin(x)\cos(y)$','Interpreter','latex');
exportgraphics(f2, 'fig_surf.png', 'Resolution', 300);
exportgraphics(f2, 'fig_surf.pdf');

% 3) Konturový graf s označením kritických křivek
f3 = figure('Color','w');
contour(xg, yg, zg, 15, 'LineWidth', 1.2); grid on; axis equal
xlabel('$x$','Interpreter','latex'); ylabel('$y$','Interpreter','latex');
title('Kontury $z=\sin(x)\cos(y)$','Interpreter','latex');
```

```
exportgraphics(f3, 'fig_contour.png', 'Resolution', 300);
exportgraphics(f3, 'fig_contour.pdf');
```

Listing 3.1: Generování grafů a export (PNG/PDF)

Proč:

- `exportgraphics` jsme používali u Dne 18 – dává kontrolu nad rozlišením (DPI) a formátem; PDF/EPS pro publikace (vektor), PNG/JPG pro prezentace.
- LaTeX popisky (`Interpreter='latex'`) zvyšují čitelnost a konzistenci symbolů.
- `figure('Color','w')` zajistí bílé pozadí (lepší tisk/komprese).

A.2 Tiled layout mini-koláž

```
f4 = figure('Color','w');
t = tiledlayout(2,2,'TileSpacing','compact','Padding','compact');

% (1) 2D křivka
nexttile; plot(x, y1, 'LineWidth', 1.8); grid on
title('sin(x)')

% (2) 2D porovnání
nexttile; plot(x, y1); hold on; plot(x, y2,'--'); grid on
legend('sin','0.5cos(2x)','Location','best'); title('Porovnání')

% (3) 3D mesh
nexttile; mesh(xg, yg, zg); title('mesh'); xlabel('x'); ylabel('y
    '); zlabel('z')

% (4) Kontury
nexttile; contourf(xg, yg, zg, 12); colorbar; title('contourf')

title(t,'Mini-koláž grafů'); % nadpis celé koláže
exportgraphics(f4, 'fig_kolaz.pdf');  % Proč: vše v jednom PDF
    pro vložení do reportu
```

Listing 3.2: Koláž do prezentace (tiledlayout) + export jedním souborem

Proč:

- `tiledlayout/nexttile` (Den 18) poskytuje čisté, konzistentní rozvržení více grafů v jednom obrázku.
- Jediný export (`exportgraphics`) usnadní vkládání do prezentace/reportu.

3.7.2 Část B – Projektový blok: Mini-aplikace (GUI + animace + práce s daty)

Cíl: v jednom okně propojit ovládací prvky (slider, tlačítka), **animaci** (harmonický oscilátor) a **práci s daty** (syntetická měření uložená ve `struct`/`cell`).

B.1 Popis mini-aplikace

- Levý panel: ovládání amplitudy a tlumení oscilátoru (`slider`).
- Horní graf: aktuální tvar křivky $y(t) = Ae^{-\alpha t}\cos(\omega t)$.
- Dolní graf: *„naměřená“* data (syntetická), možnost je znovu vygenerovat (tlačítko).
- Tlačítko *Play*: spustí animaci (kreslí se trajektorie v čase).

B.2 Kompletní kód mini-aplikace

```
% B.2: Mini-aplikace v jednom skriptu (pouze kódy z prvních 3 tý
    dnů)

% --- Data a parametry ---
Fs = 60;                    % "vzorkování" animace (fps)
t  = linspace(0, 10, Fs*10);  % 10 s
omega = 2*pi*1;             % 1 Hz

% syntetická data (struct + cell)
data = struct();
data.desc = 'Syntetické tlumené měření';
data.raw  = {};             % cell: každé měření jako vektor
data.params = [];           % záznam amplitudy a tlumení

% --- Hlavní figura a rozvržení ---
f = figure('Color','w','Name','Mini-aplikace: oscilátor','
    Position',[100 100 900 600]);
tiled = tiledlayout(f,2,2,'TileSpacing','compact','Padding','
    compact');

% --- Ovládací prvky (levý sloupec) ---
nexttile(tiled,1,[2 1]); axis off
uicontrol('Style','text','String','Amplituda A','Position',[40
    470 120 20]);
sA = uicontrol('Style','slider','Min',0,'Max',2,'Value',1,'
    Position',[40 450 120 20]);

uicontrol('Style','text','String','Tlumení \alpha','Position',[40
     400 120 20]);
sAlpha = uicontrol('Style','slider','Min',0,'Max',1,'Value',0.1,'
    Position',[40 380 120 20]);

playBtn = uicontrol('Style','pushbutton','String','Play animaci'
    ,...
     'Position',[40 320 120 30]);

regenBtn = uicontrol('Style','pushbutton','String','Nová data'
    ,...
     'Position',[40 280 120 30]);

```

```
% --- Grafy (pravý sloupec: nahoře křivka, dole data) ---
ax1 = nexttile(tiled,2); grid(ax1,'on')
title(ax1,'Okamžitá křivka y(t)')
xlabel(ax1,'t [s]'); ylabel(ax1,'y')

ax2 = nexttile(tiled,4); grid(ax2,'on')
title(ax2,'Syntetická měření')
xlabel(ax2,'t [s]'); ylabel(ax2,'y')

% --- Počáteční vykreslení ---
A = sA.Value; alpha = sAlpha.Value;
y = A*exp(-alpha*t).*cos(omega*t);
hCurve = plot(ax1, t, y, 'LineWidth',2); hold(ax1,'on')
hDot   = plot(ax1, t(1), y(1), 'ro','MarkerFaceColor','r');  % animovaný bod

% --- Funkce pro regeneraci "měření" ---
makeMeasurement = @() (A*exp(-alpha*t).*cos(omega*t) + 0.1*randn(size(t)));

% inicializace datasetu
ymeas = makeMeasurement();
data.raw{end+1} = ymeas;
data.params(end+1,:) = [A, alpha];

hMeas = plot(ax2, t, ymeas, 'Color',[0 0.4 0.8]); hold(ax2,'on')
legend(ax2, {'měření 1'}, 'Location','best')

% --- Callback: slider -> přepočet křivky ---
sA.Callback     = @(~,~) updateCurve();
sAlpha.Callback = @(~,~) updateCurve();

function updateCurve()
    A = sA.Value; alpha = sAlpha.Value;
    y = A*exp(-alpha*t).*cos(omega*t);
    set(hCurve,'YData',y);
end

% --- Callback: Nová data (syntetický) ---
regenBtn.Callback = @(~,~) regenerateData();
function regenerateData()
    A = sA.Value; alpha = sAlpha.Value;
    ymeas = A*exp(-alpha*t).*cos(omega*t) + 0.1*randn(size(t));
    data.raw{end+1} = ymeas;
    data.params(end+1,:) = [A, alpha];
    plot(ax2, t, ymeas, 'Color',[0 0.4 0.8 0.5]);
    lgd = arrayfun(@(k) sprintf('měření %d',k), 1:numel(data.raw)
        , 'UniformOutput', false);
    legend(ax2, lgd, 'Location','best');
end

```

```
% --- Callback: Play animace ---
playBtn.Callback = @(~,~) playAnim();
function playAnim()
    A = sA.Value; alpha = sAlpha.Value;
    y = A*exp(-alpha*t).*cos(omega*t);
    for k = 1:numel(t)
        set(hDot,'XData',t(k),'YData',y(k));
        drawnow;                    % okamžitá aktualizace
        pause(1/Fs);                % plynulost ~ fps
    end
end
```

Listing 3.3: Mini-aplikace: GUI + animace + data (struct/cell)

Proč:

- **GUI prvky** (`uicontrol`) a **callbacky** (`.Callback = @...`) – z Dne 20; umožní interaktivitu bez složitého App Designeru.
- **Animace** (`drawnow`, `pause`) – z Dne 19; vizualizuje průběh v čase.
- **Data jako `struct` a `cell`** – z druhého týdne; ukládáme více sérií měření a jejich parametry bez závislosti na `readtable`.
- **Tiledlayout** – z Dne 18; elegantní rozvržení ovládání a grafů.

3.7.3 Část C – 20 úloh z matematiky a fyziky (rekapitulace 3 týdnů)

Níže 20 úloh (od jednodušších po náročnější). Každá má kompaktní MATLAB kód a krátké **Proč**. Používáme pouze témata probíraná v prvních 3 týdnech: matice, vektory, skripty-/funkce, 2D/3D grafy, formátování, více grafů v okně, animace, interaktivita (`ginput`, `uicontrol`).

1. **Kinematika 1D – volný pád (graf y(t)).**

```
g = 9.81; t = linspace(0,3,200); y0 = 20; v0 = 0;
y = y0 + v0*t - 0.5*g*t.^2; y(y<0)=0;
plot(t,y,'LineWidth',2); grid on; xlabel('t'); ylabel('y');
   title('Volný pád')
```

   Proč: Základní fyzikální model a 2D vizualizace.

2. **Harmonický oscilátor – fázová křivka** $(y, \dot{y})$**.**

```
omega = 2*pi*1; t = linspace(0,10,1000);
y = cos(omega*t); v = -omega*sin(omega*t);
plot(y,v); axis equal; grid on; xlabel('y'); ylabel('v');
   title('Fázová křivka')
```

   Proč: Zobrazení stavu v prostoru $(y, \dot{y})$.

3. **Energetický potenciál** $U(x) = \frac{1}{2}kx^2$**.**

```
k = 3; x = linspace(-3,3,400); U = 0.5*k*x.^2;
plot(x,U,'LineWidth',2); grid on; xlabel('x'); ylabel('U')
```

Proč: Parabolický potenciál — přímé použití 2D plotu.

4. **Trajektorie v poli – šroubovice (3D).**

```
t = linspace(0,8*pi,800); x = cos(t); y = sin(t); z = 0.1*t;
plot3(x,y,z,'LineWidth',1.5); grid on; xlabel('x'); ylabel('y
    '); zlabel('z'); title('Šroubovice')
```

Proč: 3D křivka `plot3` (Den 17).

5. **3D povrch $z = \sin r / r$ (sinc).**

```
[x,y] = meshgrid(linspace(-8,8,200));
r = sqrt(x.^2+y.^2)+eps; z = sin(r)./r;
surf(x,y,z); shading interp; colormap turbo; colorbar
xlabel('x'); ylabel('y'); zlabel('z'); title('sinc(r)')
```

Proč: Typický hladký povrch; `shading` pro lepší vzhled.

6. **Kontury ekvipotenciálu.**

```
contour(x,y,z,20,'LineWidth',1); axis equal; grid on
title('Ekvipotenciální kontury')
```

Proč: 2D mapa hladin — rychlá orientace v poli.

7. **Vlnění na struně (časové snapshoty).**

```
x = linspace(0,1,300); k = 5*pi; c = 1;
for tau = 0:0.2:2
    y = sin(k*(x - c*tau));
    plot(x,y,'LineWidth',2); ylim([-1 1]); grid on
    title(sprintf('t = %.1f s',tau)); drawnow
end
```

Proč: Animovaná ilustrace vlny (Den 19).

8. **Balistická křivka (bez odporu).**

```
g=9.81; v0=20; ang=45*pi/180;
t = linspace(0,2*v0*sin(ang)/g,200);
x = v0*cos(ang)*t; y = v0*sin(ang)*t - 0.5*g*t.^2;
y(y<0)=0; plot(x,y,'LineWidth',2); grid on; axis equal
xlabel('x'); ylabel('y'); title('Balistická křivka')
```

Proč: Klasika 2D kinematiky.

9. **Superpozice dvou vln (interference).**

```
x = linspace(0,10,600); y = sin(2*x) + 0.7*sin(3*x+1);
plot(x,y); grid on; xlabel('x'); ylabel('y'); title('
    Superpozice vln')
```

Proč: Vizuální interference.

10. **2D pole: hladinová mapa a vektorové směry (bez `quiver`).**

```
[X,Y] = meshgrid(linspace(-2,2,60));
Z = X.*exp(-X.^2 - Y.^2);
contourf(X,Y,Z,12); colorbar; axis equal tight
title('Skalární pole Z = X e^{-(X^2+Y^2)}')
```

Proč: Interpretace skalárního pole (vizuálně).

11. **Parametrická křivka Lissajous.**

```
t = linspace(0,2*pi,1000);
x = sin(3*t); y = sin(4*t + pi/3);
plot(x,y); axis equal; grid on; title('Lissajous')
```

Proč: Periodické dráhy a rezonance.

12. **Hranice stability – tlumený oscilátor (změna α).**

```
t = linspace(0,10,1000); w = 2*pi*1; A=1; alphas = [0, 0.2,
    0.5];
for a = alphas
    y = A*exp(-a*t).*cos(w*t); plot(t,y,'LineWidth',1.5);
        hold on
end
grid on; legend('0','0.2','0.5'); xlabel('t'); ylabel('y')
title('Vliv tlumení \alpha')
```

Proč: Vizualizace tlumení v čase.

13. **3D síť (mesh) stejné funkce pro srovnání.**

```
mesh(x,y,z); xlabel('x'); ylabel('y'); zlabel('z'); title('
    sinc(r) - mesh'); grid on
```

Proč: `mesh` (Drátový model) zvýrazní reliéf.

14. **3D pohledy (otáčení scény).**

```
surf(x,y,z); shading interp; for az = 0:10:360, view(az,30);
    drawnow; end
```

Proč: Rychlý 3D „průlet“ (Den 19/17).

15. **Více grafů: porovnání variant v jednom okně.**

```
tiledlayout(2,2);
nexttile; plot(t, sin(t)); title('sin')
nexttile; plot(t, cos(t)); title('cos')
nexttile; plot(t, sin(2*t)); title('sin(2t)')
nexttile; plot(t, cos(2*t)); title('cos(2t)')
```

Proč: Přehlednost (Den 18).

16. **Interaktivní značka: klik na extrém (ginput).**

```
x = linspace(0,2*pi,400); y = sin(x).*cos(2*x);
plot(x,y); grid on; title('Klikni na lokální extrém');
[xc,yc] = ginput(1); hold on; plot(xc,yc,'ro','
   MarkerFaceColor','r')
```

Proč: Ruční výběr bodu z grafu.

17. **Animace pohybujícího se bodu po kružnici.**

```
th = linspace(0,2*pi,300); figure; axis([-1.2 1.2 -1.2 1.2]);
    axis square; grid on
hp = plot(cos(th(1)), sin(th(1)),'ro','MarkerFaceColor','r');
for k=1:numel(th), set(hp,'XData',cos(th(k)),'YData',sin(th(k
   ))); drawnow; end
```

Proč: Jednoduchý základ animace (Den 19).

18. **Porovnání měřené vs. modelové křivky.**

```
t = linspace(0,5,400); y_model = exp(-0.3*t).*sin(4*t);
y_meas  = y_model + 0.1*randn(size(t));
plot(t,y_model,'LineWidth',2); hold on; plot(t,y_meas,'.');
   grid on
legend('model','měření'); title('Model vs. data (synteticky)'
   )
```

Proč: Vizualizace shody/odchylky (bez statistik z Týdne 4).

19. **3D kontury (contour3) na povrchu.**

```
contour3(x,y,z,20,'LineWidth',1); grid on; xlabel x; ylabel y
   ; zlabel z; title('contour3')
```

Proč: Hřbetové linie v 3D – jiný pohled na stejné pole.

Poznámky k „Proč“ (obecné principy):

- **Export**: PNG (rastrový) je vhodný pro prezentace; PDF/EPS (vektorové) pro sazbu v LaTeXu.
- **Tiledlayout**: udržuje konzistentní rozvržení a mezery — vizuálně profesionální výstup.
- **Animace**: `drawnow` vykreslí okamžitě; `pause` reguluje rychlost.
- **Interaktivita**: `ginput` dovolí rychle vybrat bod; `uicontrol` umožní ovládání parametrů bez nutnosti App Designeru.
- **3D náhledy**: `view(az,el)` a `rotate3d` (volitelně) pomohou najít nejlepší záběr.

Kapitola 4

4 týden - Práce s daty

4.1 Pondělí (4. týden) – Práce s daty: Načítání CSV a Excel (readmatrix, readtable)

Níže najdeš kompletní cvičení: **20 úloh** na načítání a předzpracování dat v MATLABu. Nejprve si připrav testovací dataset (`data.csv`) a malý soubor se stanicemi (`stations.xlsx`), abys měl na čem cvičit. Poté jsou úlohy, kompaktní řešení a velmi podrobné komentáře „*Proč*“ u každého kroku.

4.1.1 Testovací data (ulož do souboru `data.csv`, kódování UTF-8)

```
Date,Time,Station,Temperature,Humidity,WindSpeed,Precip,Valid,Notes
2025-06-01,08:00:00,ST01,15.2,82,3.5,0,no,morning
2025-06-01,12:00:00,ST01,20.5,60,4.0,0,yes,noon
2025-06-01,18:00:00,ST01,18.9,70,2.5,0,yes,evening
2025-06-02,08:00:00,ST01,NA,85,3.0,0,no,sensor error
2025-06-02,12:00:00,ST01,22.1,58,4.2,0,yes,
2025-06-02,18:00:00,ST01,19.4,65,,0,yes,missing wind
2025-06-03,08:00:00,ST02,13.8,88,2.0,1,yes,rain
2025-06-03,12:00:00,ST02,16.2,80,2.5,0,yes,
2025-06-03,18:00:00,ST02,15.0,83,1.8,0,yes,
2025-06-04,12:00:00,ST03,25.5,40,5.2,0,yes,hot
2025-06-04,18:00:00,ST03,23.1,45,4.8,0,yes,
2025-06-05,12:00:00,ST01,21.0,55,3.9,0,yes,
```

Poznámka: sloupec `Temperature` obsahuje hodnotu `NA` (text) a prázdné položky pro `WindSpeed` – to je záměr, abys mohl testovat práci s chybějícími hodnotami.

4.1.2 Doporučení pro Excel soubor (`stations.xlsx`)

Vytvoř list (sheet) jménem `Stations` s těmito sloupci a hodnotami:

```
Station,Name,Latitude,Longitude,Elevation_m
ST01,Station A,50.087,14.421,200
ST02,Station B,50.075,14.437,180
ST03,Station C,50.093,14.400,220
```

Ulož jako `stations.xlsx` (Excel) – můžeš to také vygenerovat v MATLABu pomocí `writetable` (ukážu níže).

4.1.3 Úkoly (20)

1. Ulož uvedený text jako `data.csv` (UTF-8) a načti ho do MATLABu pomocí `readtable`.
2. Zobraz prvních 6 řádků (head) a informace o proměnných (rozměry, názvy sloupců).
3. Načti pouze numerická data do matice pomocí `readmatrix` a ukaž, co vrací.
4. Použij `detectImportOptions` a nakonfiguruj tak, aby `'NA'` bylo považováno za chybějící hodnotu, poté použij `readtable` s tímto `opts`.

5. Spoj sloupce `Date` a `Time` do jednoho sloupce typu `datetime` a přidej jako nový sloupec `Datetime`.
6. Převeď tabulku na `timetable` (index podle `Datetime`).
7. Pomocí `retime` spočítej denní průměry teploty.
8. Vykresli časovou řadu teplot pro každou stanici (každá stanice jinou barvu) a přidej legendu a popisky.
9. Najdi a odstraň řádky bez platné teploty (`NaN`) a ulož výsledek do `data`$_{c}$*lean.csv*.*Nahraďchybějícího*
10. Načti soubor jako `cell array` pomocí `readcell` a ukaž první řádek (hlavičku).
11. Načti list `Stations` z `stations.xlsx` a spoj (join) metadata stanic s hlavní tabulkou podle pole `Station`.
12. Vyfiltruj záznamy z intervalu 2025-06-02 až 2025-06-04 a ulož do nové tabulky.
13. Seřaď tabulku podle `Temperature` sestupně a vypiš top 5.
14. Vypočítej základní statistiky (min, max, mean, median, std) pro `Temperature` a `Humidity`.
15. Ulož (exportuj) denní průměry do CSV (`daily`$_{m}$*ean.csv*)*adoExcelu*(

4.1.4 Řešení (kompaktní MATLAB kód)

```
%% 0. (Příprava) - pokud nemáš stations.xlsx, vytvoříme jej z MATLABu
Stations = table({'ST01';'ST02';'ST03'}, {'Station A';'Station B';'Station C'}, ...
    [50.087;50.075;50.093], [14.421;14.437;14.400], [200;180;220], ...
    'VariableNames', {'Station','Name','Latitude','Longitude','Elevation_m'});
writetable(Stations,'stations.xlsx','Sheet','Stations');

%% 1. Načtení CSV pomocí readtable
T = readtable('data.csv');

%% 2. Zobraz prvních 6 řádků a info o proměnných
disp(T(1:6,:))
summary(T)
disp(['Velikost tabulky: ' num2str(size(T,1)) ' x ' num2str(size(T,2))])

%% 3. Načtení numerických dat pomocí readmatrix
M = readmatrix('data.csv');

%% 4. detectImportOptions + TreatAsMissing='NA'
opts = detectImportOptions('data.csv');
opts = setvaropts(opts, 'Temperature', 'TreatAsMissing', {'NA'});
```

```
T2 = readtable('data.csv', opts);

%% 5. Spojení Date+Time -> Datetime
T2.Datetime = datetime(strcat(string(T2.Date),' ',string(T2.Time)), 'InputFormat','yyyy-MM-dd HH:mm:ss');

%% 6. Převod na timetable
TT = table2timetable(T2,'RowTimes','Datetime');

%% 7. Denní průměry teplot
TTnum = TT(:, {'Temperature'}); % zachovej jen numerické proměnné
daily = retime(TTnum, 'daily', 'mean');

%% 8. Časová řada teplot pro každou stanici
stations = unique(T2.Station);
figure;
hold on;
for i=1:numel(stations)
    idx = strcmp(T2.Station, stations{i});
    plot(T2.Datetime(idx), T2.Temperature(idx), '-o','DisplayName',stations{i});
end
hold off; legend('Location','best'); xlabel('Datetime'); ylabel('Temperature');

%% 9. Odstranění řádků bez platné teploty
Tclean = rmmissing(T2, 'DataVariables', 'Temperature');
writetable(Tclean,'data_clean.csv');

%% 10. Interpolace chybějících WindSpeed hodnot
% Předpoklad: WindSpeed numerické, chybějící jako NaN
T2.WindSpeed = str2double(string(T2.WindSpeed)); % pokud byly texty -> NaN
T2.WindSpeed = fillmissing(T2.WindSpeed,'linear');

%% 11. Načtení jako cell array
C = readcell('data.csv');

%% 12. Načtení stations.xlsx a spojení
S = readtable('stations.xlsx','Sheet','Stations');
Tjoined = innerjoin(T2, S, 'Keys', 'Station');

%% 13. Filtrace dle datumu
tfilt = (T2.Datetime >= datetime(2025,6,2)) & (T2.Datetime <= datetime(2025,6,4));
T_range = T2(tfilt,:);

%% 14. Seřazení podle teploty (sestupně) a top 5
T_sorted = sortrows(T2,'Temperature','descend');
top5 = T_sorted(1:min(5,height(T_sorted)), :);

```

```
%% 15. Základní statistiky
temp_stats.min = min(T2.Temperature,'omitnan');
temp_stats.max = max(T2.Temperature,'omitnan');
temp_stats.mean = mean(T2.Temperature,'omitnan');
temp_stats.median = median(T2.Temperature,'omitnan');
temp_stats.std = std(T2.Temperature,'omitnan');

%% 16. Export denních průměrů
daily_table = timetable2table(daily,'RowTimes','Date');
writetable(daily_table,'daily_mean.csv');
writetable(daily_table,'daily_mean.xlsx','Sheet','DailyMean');

%% 17. Načtení vybraných sloupců (pomocí opts)
opts2 = detectImportOptions('data.csv');
opts2.SelectedVariableNames = {'Date','Time','Station','
    Temperature'};
T_sel = readtable('data.csv', opts2);

%% 18. Převod Station a Notes na string / categorical
T2.Station = string(T2.Station);
T2.Notes   = string(T2.Notes);
T2.Station_cat = categorical(T2.Station);

%% 19. groupsummary / varfun - průměr a počet záznamů podle
    stanice
G = groupsummary(T2,'Station','mean','Temperature');
Count = groupsummary(T2,'Station','numel','Temperature');

%% 20. Vykreslení denních průměrů a export PNG 300 DPI
f = figure('Color','w'); plot(daily.Time, daily.Temperature,'-o',
    'LineWidth',1.6);
xlabel('Den'); ylabel('Průměrná teplota [ C ]'); title('Denní prů
    měry teplot');
grid on;
exportgraphics(f,'daily_mean_plot.png','Resolution',300);
```

4.1.5 Řešení s podrobným vysvětlením (řádek po řádku, Proč)

Níže rozepisuji každou úlohu s komentovaným kódem a vysvětlením, proč je krok proveden a jaké jsou běžné potíže.

1. **Uložení `data.csv` a načtení pomocí `readtable`**

```
T = readtable('data.csv');
```

Proč:

- `readtable` je výchozí volba pro načítání heterogenních dat (čísla + text + prázdné buňky). Vrací objekt typu `table`, který má pojmenované sloupce (proměnné) a je velmi pohodlný pro indexování, filtrování a analýzu.

- `readtable` se snaží odhadnout typy sloupců a správně naimportovat data; pokud máš speciální formát (oddělovač, kódování), přidej jmenné argumenty.

2. **Zobrazení prvních řádků a informace o proměnných**

```
disp(T(1:6,:))
summary(T)
disp(['Velikost tabulky: ' num2str(size(T,1)) ' x ' num2str(
    size(T,2))])
```

Proč:

- `T(1:6,:)` vytiskne prvních 6 řádků — rychlé „head“.
- `summary(T)` poskytne přehled proměnných, typů a počtu chybějících hodnot.
- Kontrola rozměrů (`size`) potvrzuje, že jsme načetli očekávaný počet řádků/sloupců.

3. **Načtení číselných dat pomocí `readmatrix`**

```
M = readmatrix('data.csv');
```

Proč:

- `readmatrix` čte z CSV pouze hodnoty, které lze interpretovat jako čísla. Vrátí číselnou matici (přičemž místo nenačitatelných hodnot vloží `NaN`).
- Je užitečné, když potřebuješ rychle pracovat s numerickou částí dat (matematické operace, linear algebra) bez metadata sloupců.
- Nevýhoda: ztratíš názvy sloupců a sloupce s textem.

4. **Použití `detectImportOptions` a ošetření `'NA'` jako missing**

```
opts = detectImportOptions('data.csv');
opts = setvaropts(opts, 'Temperature', 'TreatAsMissing', {'NA
    '});
T2 = readtable('data.csv', opts);
```

Proč:

- `detectImportOptions` vrátí objekt `opts` s navrženými pravidly importu (oddělovač, názvy proměnných, typy).
- Nastavením `TreatAsMissing` říkáme importní proceduře, že text `'NA'` má být považován za chybějící hodnotu (`NaN`).
- Poté `readtable(...,opts)` provede načtení podle těchto pravidel — lépe kontroluje typy a missing hodnoty.

5. **Spojení `Date` + `Time` do `datetime`**

```
T2.Datetime = datetime(strcat(string(T2.Date),' ',string(T2.
    Time)), ...
                       'InputFormat','yyyy-MM-dd HH:mm:ss');
```

Proč:

- Často jsou datum a čas ve dvou sloupcích; pro časové analýzy je efektivnější jeden `datetime` sloupec.
- `strcat` spojuje texty, `string(...)` zaručí konzistenci (např. pokud některý sloupec je cell array).
- `InputFormat` zadává přesný formát, což zrychlí parsování a sníží riziko chyb.

6. **Převod na `timetable`**

```
TT = table2timetable(T2,'RowTimes','Datetime');
```

Proč:

- `timetable` používáme, když pracujeme s časovými řadami — umožňuje funkce jako `retime`, zarovnávání, resampling apod.
- `table2timetable` vytvoří časový index z pole `Datetime` a zachová ostatní proměnné jako sloupce.

7. **Denní průměry přes `retime`**

```
TTnum = TT(:, {'Temperature'});     % zachovej jen číselné
   proměnné
daily = retime(TTnum, 'daily', 'mean');
```

Proč:

- `retime` provádí agregaci časových dat — zde mění časovou mřížku na denní a aplikuje průměr.
- Nejprve vybereme pouze numerické sloupce (jinak `retime` nemusí fungovat).
- Výsledek je timetable s indexem (každý den) a průměrem teplot.

8. **Plot teplot pro každou stanici (oddělené barvy)**

```
stations = unique(T2.Station);
figure; hold on;
for i=1:numel(stations)
    idx = strcmp(T2.Station, stations{i});
    plot(T2.Datetime(idx), T2.Temperature(idx), '-o','
       DisplayName',stations{i});
end
hold off; legend('Location','best'); xlabel('Datetime');
   ylabel('Temperature');
```

Proč:

- `unique` získá seznam stanic. Smyčka vykreslí jednu čáru na stanici.
- `strcmp` vytváří masku pro záznamy dané stanice.
- `DisplayName` a `legend` vytvářejí přehlednou legendu.

9. **Odstranění řádků bez platné teploty**

```
Tclean = rmmissing(T2, 'DataVariables', 'Temperature');
writetable(Tclean,'data_clean.csv');
```

Proč:

- `rmmissing` odstraní řádky, kde je `Temperature` chybějící (`NaN`).
- Po vyčištění data ukládáme pomocí `writetable` pro další zpracování.

10. **Interpolace chybějících `WindSpeed`**

```
T2.WindSpeed = str2double(string(T2.WindSpeed)); % zajištění, že jsou čísla
T2.WindSpeed = fillmissing(T2.WindSpeed,'linear');
```

Proč:

- Pokud `WindSpeed` obsahuje prázdné textové buňky, převedeme je na `NaN` pomocí `str2double`.
- `fillmissing(...,'linear')` provede lineární interpolaci přes NaN hodnoty — vhodné, když data jsou v časové či jiné přirozené posloupnosti.
- U chronologických dat je vhodné zajistit, že jsou řádky seřazeny podle času před interpolací.

11. **Načtení jako `cell array`**

```
C = readcell('data.csv');
```

Proč:

- `readcell` vrátí syrový cell array — často se hodí pro prohlížení přesně toho, co je v buňkách (text, čísla, prázdné).
- První řádek (hlavička) je obvykle cell s názvy sloupců; zkontroluj si je `C(1,:)`.

12. **Načíst `stations.xlsx` a provést spojení s metadaty**

```
S = readtable('stations.xlsx','Sheet','Stations');
Tjoined = innerjoin(T2, S, 'Keys', 'Station');
```

Proč:

- `readtable` umí číst Excel soubory; parametr `'Sheet'` zvolí konkrétní list.
- `innerjoin` spojí dvě tabulky podle shodných hodnot ve sloupci `Station`. Výsledek obsahuje jen řádky, které mají odpovídající záznam v obou tabulkách.
- Alternativy: `join` (levé připojení), `outerjoin` (plné).

13. **Filtrace dle datumu 2025-06-02 až 2025-06-04**

```
tfilt = (T2.Datetime >= datetime(2025,6,2)) & (T2.Datetime <= datetime(2025,6,4));
T_range = T2(tfilt,:);
```

Proč:

- Logické masky jsou nejpohodlnější způsob filtrování tabulek.

- `datetime(YYYY,MM,DD)` vytváří pevný bod v čase; rozsah pak vybere řádky v intervalu.

14. **Seřazení podle `Temperature` sestupně a top 5**

```
T_sorted = sortrows(T2,'Temperature','descend');
top5 = T_sorted(1:min(5,height(T_sorted)), :);
```

Proč:

- `sortrows` řadí tabulku podle zvoleného sloupce; parametr `'descend'` dává sestupné pořadí.
- `height` vrátí počet řádků; `min` zabezpečí, že nesáhneš přes konec tabulky.

15. **Základní statistiky pro `Temperature` a `Humidity`**

```
temp_stats.min = min(T2.Temperature,'omitnan');
temp_stats.max = max(T2.Temperature,'omitnan');
temp_stats.mean = mean(T2.Temperature,'omitnan');
temp_stats.median = median(T2.Temperature,'omitnan');
temp_stats.std = std(T2.Temperature,'omitnan');
```

Proč:

- Volba `'omitnan'` zajistí, že chybějící hodnoty nejsou počítány.
- Tento soubor statistik je typickým výstupem při prvotní analýze dat.

16. **Export denních průměrů do CSV a Excelu**

```
daily_table = timetable2table(daily,'RowTimes','Date');
writetable(daily_table,'daily_mean.csv');
writetable(daily_table,'daily_mean.xlsx','Sheet','DailyMean')
    ;
```

Proč:

- `timetable2table` převede `timetable` na běžný `table` s datovým sloupcem (zpravidla sloupec `Date`).
- `writetable` vytváří CSV nebo Excel — vhodné pro sdílení výsledků s kolegy, import do Excelu nebo další část pipeline.

17. **Načtení jen vybraných sloupců**

```
opts2 = detectImportOptions('data.csv');
opts2.SelectedVariableNames = {'Date','Time','Station','
    Temperature'};
T_sel = readtable('data.csv', opts2);
```

Proč:

- Pokud máš velký soubor a potřebuješ pouze několik sloupců, zrychlí to načítání a snižuje paměťovou stopu.

- `SelectedVariableNames` je pohodlný způsob, jak importovat jen to, co potřebuješ.

18. **Převod `Station` a `Notes` na `string` / `categorical`**

```
T2.Station = string(T2.Station);
T2.Notes   = string(T2.Notes);
T2.Station_cat = categorical(T2.Station);
```

Proč:

- `string` poskytuje moderní práci s textem (vektorový typ, funkce jako `contains`, `strlength` apod.).
- `categorical` je vhodné, když máš omezený počet kategorií (stanice): šetří paměť a urychluje skupinové operace (např. `groupsummary`).
- Rozdíl: `string` je text; `categorical` reprezentuje nominální proměnné s omezenou množinou hodnot.

19. **`groupsummary`/`varfun`: průměry a počty podle stanice**

```
G = groupsummary(T2,'Station','mean','Temperature');
Count = groupsummary(T2,'Station','numel','Temperature');
```

Proč:

- `groupsummary` je rychlý způsob, jak získat agragované statistiky podle skupin.
- Alternativně lze použít `varfun` nebo `findgroups`/`splitapply` pro větší flexibilitu.

20. **Vykreslení denních průměrů a export 300 DPI PNG**

```
f = figure('Color','w');
plot(daily.Time, daily.Temperature,'-o','LineWidth',1.6);
xlabel('Den'); ylabel('Průměrná teplota [ C ]'); title('Denní
    průměry');
grid on;
exportgraphics(f,'daily_mean_plot.png','Resolution',300);
```

Proč:

- `exportgraphics` umožní regulovat rozlišení (DPI) a formát — 300 DPI je dobrý standard pro tisk/prezentace.
- Bílé pozadí a tlustší čára zlepšují čitelnost v projekci a tisku.

4.1.6 Nové příkazy / konstrukty použité v řešeních (poznámky a vysvětlení)

- `detectImportOptions(filename)` Vrací objekt s nastavením importu (oddělovač, názvy sloupců, navržené typy proměnných). Použití: upravíš `opts` před zavoláním `readtable`.

- `setvaropts(opts, varname, 'TreatAsMissing', {...})` Umožňuje říct, jaké texty považovat za chybějící (`'NA'`, `''`, atd.). Velmi užitečné pro nekonzistentní CSV.
- `readmatrix` vs. `readtable` vs. `readcell` `readmatrix` — výstup číselná matice (NaN pro nečíselné). `readtable` — vrací `table` (nazvané sloupce, heterogenní typy). `readcell` — syrový cell array (obsahuje texty i čísla).
- `table2timetable` a `retime` `table2timetable` vytvoří časově indexovaná data; `retime` agreguje/resampluje (např. denní průměry).
- `fillmissing` a `rmmissing` Standardní metody pro ošetření chybějících hodnot (interpolace, vymazání). Volby: `'linear'`, `'previous'`, `'next'`, `'constant'`.
- `groupsummary`, `varfun`, `findgroups`/`splitapply` Skupinové operace nad tabulkami — velmi užitečné pro agregační analýzy podle kategorie.
- `exportgraphics` Moderní způsob exportu figure do PNG/PDF s nastavením rozlišení (DPI) a vektorového obsahu pro PDF.

4.1.7 Doplňkové tipy (best practices)

- Při práci s časovými daty vždy ověř, že časová osa je unikátní a seřazená; pro agregace to výrazně zjednoduší práci.
- Před interpolací seřaď záznamy podle času a pozor na hranice (extrapolace nemusí dávat smysl).
- Používej `detectImportOptions` pro velké nebo nekonzistentní CSV — usnadní to práci s typy a chybějícími hodnotami.
- Při exportu grafů do publikací preferuj vektorové formáty (PDF/EPS) pro čitelnost textu při škálování; pro prezentace zase PNG s DPI 300.

4.2 Úterý – 4. týden: Filtrace dat podle podmínek

Níže najdeš kompletní úterní materiál: nejprve ukázkový soubor `students.csv` (ČEŠTINA, 25 záznamů), poté 20 úloh s kompaktním MATLAB kódem a velmi podrobnými, krok-za-krokem vysvětleními (PROČ). Vše v konzistentním stylu jako u předchozích dnů.

4.2.1 Ukázkový soubor `students.csv` (ulož jako UTF-8)

```
Jméno,Věk,Výška,Váha,Známka
Jan Novák,21,180,78,2
Petra Svobodová,19,168,62,1
Lukáš Procházka,23,182,80,3
Martina Dvořáková,22,170,65,2
Tomáš Svoboda,24,175,85,4
Kateřina Novotná,20,165,58,1
Pavel Kučera,26,188,90,2
Lucie Králová,18,160,55,1
Marek Horák,21,177,77,3
Jana Míková,22,172,68,2
Ondřej Němec,25,183,82,3
Eva Bartošová,19,158,54,2
Michal Pokorný,20,179,76,3
Alena Veselá,23,169,63,2
David Krejčí,24,185,88,4
Hana Štěpánková,21,166,59,2
Roman Sedláček,22,181,80,3
Barbora Hrušková,20,162,57,1
Karel Holub,27,190,92,3
Monika Černá,23,167,60,2
Josef Vacek,18,174,70,3
Kristýna Poláková,24,171,66,2
Adam Marek,22,176,75,2
Iva Říhová,25,164,56,4
Václav Blažek,21,178,79,3
```

Poznámka: ulož soubor přesně tak, aby čárky byly oddělovačem (CSV), kódování UTF-8 (aby diakritika v jménech byla v pořádku).

4.2.2 Úkoly a řešení (20)

1. **Načti `students.csv` do MATLABu pomocí `readtable`.**

```
% 1) Načtení CSV do tabulky
T = readtable('students.csv', 'TextType', 'string');
```

Vysvětlení / PROČ (krok po kroku):

(a) Používáme `readtable`, protože soubor obsahuje heterogenní data (text + čísla). `readtable` vrátí `table` s názvy sloupců odpovídajícími hlavičce CSV.

(b) Parametr `'TextType','string'` zajistí, že textová pole (Jméno) budou typu `string` místo starého `cell`—snazší práce s textovými operacemi (např. `startsWith`, `contains`).

(c) Po provedení příkazu zkontroluj proměnnou `T` v Workspace — měla by mít 25 řádků a 5 sloupců.

2. **Zobraz prvních 6 řádků a stručnou statistiku (head + summary).**

```
% 2) Zobrazení prvních řádků a shrnutí
disp(T(1:6,:));
summary(T)
```

Vysvětlení / PROČ:

(a) `T(1:6,:)` vytiskne prvních 6 řádků tabulky — rychlá kontrola, že import proběhl korektně (čtení hlaviček, oddělovač, správná čísla).

(b) `summary(T)` zobrazí typy sloupců, minimální/maximální, počet unikátních hodnot u textu, počet NaN atd. Je to základní diagnostika dat před zpracováním.

3. **Ujisti se, že sloupce mají správné datové typy; pokud ne, převeď `Jméno` na `string`, `Věk`, `Výška`, `Váha`, `Známka` na `double`.**

```
% 3) Kontrola a převod typů
% Zjistíme typy:
varTypes = varfun(@class, T, 'OutputFormat', 'cell');
disp(varTypes);

% Převedeme explicitně (bez újmy pokud jsou již ve správném
   typu)
T.Jméno   = string(T.Jméno);        % text jako string
T.Věk     = double(T.Věk);          % věk numericky
T.Výška   = double(T.Výška);        % cm numericky
T.Váha    = double(T.Váha);         % kg numericky
T.Známka  = double(T.Známka);       % grade numericky
```

Vysvětlení / PROČ:

(a) `varfun(@class,...)` zjistí aktuální typy všech sloupců; je dobré to provést pro bezpečnost (někdy při importu MATLAB označí jako `cell` nebo `char`).

(b) Převodem na `string` a `double` si zaručíme jednotné datové typy pro následné operace: porovnávání textu přes `string` je jednodušší; numerické sloupce mohou být okamžitě použity v aritmetice.

(c) Příkazy jsou idempotentní (pokud sloupec už je správného typu, převod ho nezničí).

4. **Filtrovat: vyber všechny studenty starší než 22 let.**

```
% 4) Filtrace podle věku > 22
mask_age = T.Věk > 22;
T_age_above22 = T(mask_age, :);
disp(T_age_above22);
```

Vysvětlení / PROČ:

(a) `T.Věk > 22` vytvoří logický vektor (masku) stejné délky jako tabulka; v místě, kde je podmínka pravdivá, bude `true`.

(b) Indexací `T(mask`$_{a}ge, :)vyberemepouzeřádky, kterésplňujípodmínku.Použitímaskyjeefektivní$: $žádnésmyčky, vMATLAButoběžívevektorové/kompaktníformě.$

(c) **Filtrovat: vyber studenty s výškou menší než 170 cm.**

```
% 5) Filtrace podle výšky < 170 cm
mask_short = T.Výška < 170;
T_short = T(mask_short, :);
disp(T_short);
```

Vysvětlení / PROČ:

(a) Logická maska `T.Výška < 170` vybere všechny řádky, kde je podmínka splněná.

(b) Užitečné pro selektivní statistiky (např. porovnání váhy u nižších vs. vyšších studentů).

6. **Filtrovat: vyber studenty s nejlepším hodnocením (Známka == 1).**

```
% 6) Nejlepší studenti (Známka == 1)
mask_best = T.Známka == 1;
T_best = T(mask_best,:);
disp(T_best);
```

Vysvětlení / PROČ:

(a) Použití `==` vrátí `true` pro přesnou shodu.

(b) Pozor: při práci s plovoucími typy (float) je potřeba být opatrný; u celých čísel (známky) je `==` v pořádku.

7. **Kombinovaná podmínka (AND): vyber studenty starší než 20 let *a* se známkou ≤ 2.**

```
% 7) AND: věk > 20 a známka <= 2
mask_and = (T.Věk > 20) & (T.Známka <= 2);
T_and = T(mask_and,:);
disp(T_and);
```

Vysvětlení / PROČ:

(a) Operátor element-wise `&` spojuje dvě logické masky — výsledkem je maska, která je pravdivá pouze tam, kde jsou pravdivé obě podmínky.

(b) Závorky kolem jednotlivých podmínek jsou nutné, protože precedence operátorů by mohla vést k neočekávaným výsledkům.

8. **Kombinovaná podmínka (OR): vyber studenty, kteří mají výšku $<$ 160 cm *nebo* váhu $<$ 60 kg.**

```
% 8) OR: výška < 160 OR váha < 60
mask_or = (T.Výška < 160) | (T.Váha < 60);
T_or = T(mask_or,:);
disp(T_or);
```

Vysvětlení / PROČ:

(a) Operátor | (element-wise OR) vybere řádky, kde platí alespoň jedna z podmínek.

(b) Praktické při hledání „malých / lehkých“ studentů pro specifická porovnání.

9. **Hledání podle jména: vyber studenty, jejichž jméno začíná na písmeno 'M' (příp. 'm') pomocí `startsWith`.**

```
% 9) Names starting with 'M' (case-insensitive)
mask_M = startsWith(T.Jméno, 'M', 'IgnoreCase', true);
T_M = T(mask_M,:);
disp(T_M);
```

Vysvětlení / PROČ:

(a) `startsWith` pracuje s `string` a má užitečný parametr `'IgnoreCase'`.

(b) Používáme ho k textovým filtrům (filtrace podle jména, příjmení, příznaků atd.). Je robustní vůči diakritice a velikosti písmen při správném nastavení.

10. **Převeď `Známka` na `categorical` a spočítej počet studentů v každé kategorii.**

```
% 10) Categorical pro Známka + counts
T.Známka_cat = categorical(T.Známka);
tab = countcats(T.Známka_cat);
cats = categories(T.Známka_cat);
table(cats, tab)
```

Vysvětlení / PROČ:

(a) `categorical` je vhodné pro diskrétní škály (známky 1..5): umožňuje rychlé agregace, grafy, a ušetří paměť oproti stringům, pokud je málo kategorií.

(b) `countcats` spočítá počet výskytů každé kategorie; `categories` vrátí seznam kategorií.

(c) Výsledek dá přehled, kolik studentů má kterou známku.

11. **Pro filtr (věk > 20) spočítej průměrnou výšku a váhu; použij `mean(...,'omitnan')`.**

```
% 11) Průměrná výška a váha pro věk > 20
mask = T.Věk > 20;
mean_height = mean(T.Výška(mask), 'omitnan');
mean_weight = mean(T.Váha(mask), 'omitnan');
disp(['Mean height: ', num2str(mean_height)]);
disp(['Mean weight: ', num2str(mean_weight)]);
```

Vysvětlení / PROČ:

(a) Logická maska selektuje požadovanou podmnožinu.

(b) `mean(...,'omitnan')` ignoruje případné NaN v datech (pokud by existovaly).

(c) Dává informační souhrn pro podskupinu studentů.

12. **Seřaď tabulku podle `Známka` vzestupně a potom podle `Věk` vzestupně (tie-breaker).**

```
% 12) Sort by grade (asc) then by age (asc)
T_sorted = sortrows(T, {'Známka', 'Věk'}, {'ascend','ascend'
    });
disp(T_sorted(1:10,:)); % zobraz prvních 10 pro kontrolu
```

Vysvětlení / PROČ:

(a) `sortrows` akceptuje vícenásobné klíče a orientaci (ascend/descend).

(b) To je užitečné pokud chceme např. „nejlepší studenti, mladší přednostně“.

(c) Kontrolujeme výstup prvních 10 řádků, abychom ověřili pořadí.

13. **Vyber top 5 nejtěžších studentů (podle `Váha`).**

```
% 13) Top 5 nejtěžších
T_by_weight = sortrows(T, 'Váha', 'descend');
top5_heavy = T_by_weight(1:min(5,height(T_by_weight)), :);
disp(top5_heavy);
```

Vysvětlení / PROČ:

(a) Seřadíme sestupně podle váhy a vezmeme první 5 řádků.

(b) Použití `min(...)` zabezpečí, že příkaz funguje i když je méně než 5 studentů.

14. **Vypočítej BMI pro každého studenta (BMI = váha / (výška[m])^2)*apřidejjakonovýsloupec***

```
% 14) Výpočet BMI a přidání sloupce
height_m = T.Výška / 100;              % převod cm -> m
T.BMI = T.Váha ./ (height_m.^2);     % element-wise dělení a
    umocnění
disp(T(:, {'Jméno','Výška','Váha','BMI'}));
```

Vysvětlení / PROČ:

(a) Převod na metry je zásadní (výška v cm → m).

(b) Používáme `./` a `.^`*proelement−wiseoperace*¯*každýřádekspočítámesamostatně.Přidánímsloupce*`T`

15. **Vyber studenty s BMI ≥ 25 (nadváha).**

```
% 15) Filtrace podle BMI >= 25
mask_over25 = T.BMI >= 25;
T_overweight = T(mask_over25, :);
disp(T_overweight(:, {'Jméno','BMI'}));
```

Vysvětlení / PROČ:

(a) Podmínka `T.BMI >= 25` vybere „nadváhu“ podle běžné WHO hranice.

(b) Tato filtrace je běžná v epidemiologii/biometrii.

16. **Vytvoř věkové intervaly (bins) pomocí `discretize`: 18–19, 20–21, 22–23, 24–27 a spočítej počet studentů v každém binu.**

```
% 16) Discretize ages
edges = [17.5, 19.5, 21.5, 23.5, 27.5]; % hrany (open/closed
   dle defaultu)
age_groups = discretize(T.Věk, edges);
% Počty v každé kategorii:
counts = accumarray(age_groups(~isnan(age_groups))', 1);
% Labely pro přehled:
labels = {'18-19','20-21','22-23','24-27'};
table(labels', counts)
```

Vysvětlení / PROČ:

(a) `discretize` mapuje každou věkovou hodnotu na index intervalu (bin) podle hran.

(b) `accumarray` spočítá frekvence (zde použito s hodnotou 1 pro každý případ).

(c) Výsledek je přehled rozložení věku v kategorii — užitečné pro demografii.

17. **Označ „rizikové“ studenty (známka ≥ 4) přidáním logického sloupce `AtRisk`.**

```
% 17) Označení AtRisk pro známky >= 4
T.AtRisk = T.Známka >= 4;
disp(T(:, {'Jméno','Známka','AtRisk'}));
```

Vysvětlení / PROČ:

(a) Přidání logického sloupce usnadní pozdější filtrování nebo report (např. „poslat upozornění“).

(b) Logické sloupce jsou efektivní pro boolean masky a snadno se agregují (např. `sum(T.AtRisk)` dá počet rizikových studentů).

18. **Ulož filtrovanou tabulku (např. studenti s BMI >=25) do souboru `students`$_o$*verweight.csv*.**

```
% 18) Uložení filtrované tabulky do CSV
writetable(T_overweight, 'students_overweight.csv');
```

Vysvětlení / PROČ:

(a) `writetable` uloží `table` do CSV; hodnoty typu `string` jsou správně exportovány.

(b) Tento krok je praktický, pokud chceš výsledky sdílet nebo zpracovat v jiném nástroji (Excel, Python).

19. Najdi index(y) studenta jménem `'Jan Novák'`; pokud existuje více Janů, vyber všechny.

```
% 19) Najít indexy pro konkrétní jméno
idx_jan = find(T.Jméno == "Jan Novák");
disp(['Indexy Jan Novák: ', num2str(idx_jan')]);
T(idx_jan, :)
```

Vysvětlení / PROČ:

(a) **`T.Jméno == "Jan Novák"` porovnává stringy element-wise; vrací logickou masku.**

(b) **`find` převádí masku na indexy (pozice v tabulce). Pokud jméno není, `find` vrátí prázdný vektor.**

(c) **Dále můžeme pracovat s těmito indexy (upravit, mazat, aktualizovat).**

20. Vytvoř náhodný výběr 5 studentů (bez opakování) a zobraz jejich záznamy; použi `randperm`.

```
% 20) Náhodný výběr 5 studentů
n = height(T);
k = min(5, n);
rnd_idx = randperm(n, k);
T_rand5 = T(rnd_idx, :);
disp(T_rand5);
```

Vysvětlení / PROČ:

(a) **`randperm(n,k)` vytvoří náhodný permutační vektor délky `k` bez opakování z řad 1..n.**

(b) **Používá se pro náhodný výběr (sampling) bez opakování — užitečné při tvorbě testovacích podmnožin nebo A/B experimentů.**

4.2.3 Dodatečné poznámky a tipy (krok-za-krokem rady)

- **Kontrola konzistence:** Po každé filtraci si vždy zkontroluj počet řádků (`height(...)`) a několik prvních nebo posledních záznamů (`head(...)` nebo `T(1:5,:)`), abys ověřil, že maska funguje správně.
- **Null / NaN hodnoty:** Pokud očekáváš chybějící hodnoty, použij `ismissing` nebo `rmmissing` před filtrováním, nebo pokryj jejich ošetření při výpočtech (`'omitnan'`).
- **Textové filtry:** kromě `startsWith` existuje `contains` (shoda podřetězce), `endsWith` (konec řetězce); všechny pracují s `string` a mají parametr `'IgnoreCase'`.
- **Efektivita:** Všechny filtrovací operace jsou v MATLABu vektorové a velmi rychlé pro dataset velikosti stovek až desetitisíců záznamů. Pro miliony záznamů zvaž `timetable`/`tall` nebo databázi.
- **Ukládání mezivýsledků:** Po náročnějších úpravách (čistění, interpolace) si ulož mezivýsledek do nového CSV (`writetable`) — je to jednoduchá záloha.

4.3 Středa – Výpočty statistik (mean, std, median, max, min)

V tomto bloku se zaměříme na výpočty základních statistik nad daty ze souboru `students.csv`. Použijeme funkce `mean`, `std`, `median`, `max`, `min`. Statistiky budeme počítat pro celé sloupce, i pro vyfiltrované podmnožiny.

4.3.1 Data

Pracujeme se souborem `students.csv`, který obsahuje sloupce:

```
Jméno,Věk,Výška,Váha,Známka
...
```

4.3.2 Úlohy a řešení

1. **Průměrný věk všech studentů**

```
T = readtable("students.csv");
meanAge = mean(T.Věk);
```

Proč: Funkce `mean` počítá aritmetický průměr. Používáme přímo sloupec `T.Věk`. Všichni studenti jsou zahrnuti.

2. **Medián výšky**

```
medianHeight = median(T.Výška);
```

Proč: Medián je prostřední hodnota seřazených dat. Oproti průměru je odolný vůči extrémům.

3. **Směrodatná odchylka hmotnosti**

```
stdWeight = std(T.Váha);
```

Proč: `std` měří rozptýlení kolem průměru. Čím větší, tím jsou data variabilnější.

4. **Maximum věku**

```
maxAge = max(T.Věk);
```

Proč: `max` vybere největší hodnotu vektorového sloupce. Získáme věk nejstaršího studenta.

5. **Minimum výšky**

```
minHeight = min(T.Výška);
```

Proč: `min` vrátí nejmenší hodnotu, zde tedy nejnižší student.

6. **Průměrná známka všech studentů**

```
meanGrade = mean(T.Známka);
```

Proč: Průměr ze sloupce známek vyjadřuje celkový studijní průměr skupiny.

7. **Medián známky (nejčastější střední hodnota)**

```
medianGrade = median(T.Známka);
```

Proč: Umožňuje zjistit „typickou“ známku, méně citlivou na extrémy než průměr.

8. **Rozptyl věku (nová funkce)**

```
varAge = var(T.Věk);
```

Proč: `var` dává rozptyl, tedy druhý moment kolem průměru. Čím větší, tím větší rozdíly ve věku. (Nová funkce, příbuzná `std`.)

9. **Průměrná výška studentů mladších 21 let**

```
young = T(T.Věk < 21,:);
meanHeightYoung = mean(young.Výška);
```

Proč: Nejprve logicky vyfiltrujeme podmnožinu tabulky, pak počítáme průměr jen na ní.

10. **Průměrná váha studentů se známkou 1**

```
excellent = T(T.Známka == 1,:);
meanWeightExcellent = mean(excellent.Váha);
```

Proč: Podmínka vybírá jen ty s nejlepší známkou, na nich počítáme průměr váhy.

11. **Maximum výšky u studentů s věkem > 23**

```
older = T(T.Věk > 23,:);
maxHeightOlder = max(older.Výška);
```

Proč: Kombinace filtrace a funkce `max` ukazuje, jak vybrat extrémy jen v části dat.

12. **Minimum váhy u studentů se známkou horší než 3**

```
weak = T(T.Známka > 3,:);
minWeightWeak = min(weak.Váha);
```

Proč: Ukazuje, jak podmínka na známku umožní zkoumat vlastnosti slabších studentů.

13. **Směrodatná odchylka výšky studentek (nový filtr podle jména)**

```
femaleMask = endsWith(T.Jméno,"a");
stdHeightGirls = std(T.Výška(femaleMask));
```

Proč: Používáme textovou podmínku – jména končící na „a“ označují dívky. Nová funkce `endsWith`.

14. **Průměrný věk podle známky 1 nebo 2**

```
good = T(T.Známka <= 2,:);
meanAgeGood = mean(good.Věk);
```

Proč: Podmínka `<=2` slučuje známku 1 i 2. Výpočet průměru pak shrnuje věk „lepších“ studentů.

15. **Průměrné statistiky (věk, výška, váha) všech studentů**

```
avgStats = [mean(T.Věk), mean(T.Výška), mean(T.Váha)];
```

Proč: Lze spočítat více statistik najednou a uložit je do vektoru. To je základ pro další analýzu.

16. **Medián výšky studentů se známkou 1–3**

```
midGroup = T(T.Známka <= 3,:);
medianHeightMid = median(midGroup.Výška);
```

Proč: Medián na části dat ukazuje typickou hodnotu u průměrných až dobrých studentů.

17. **Maximální váha u studentů do 20 let**

```
young20 = T(T.Věk <= 20,:);
maxWeight20 = max(young20.Váha);
```

Proč: Podmínka kombinuje věkový filtr s výpočtem maxima.

18. **Rozdíl průměrné výšky dívek a chlapců**

```
girls = T(endsWith(T.Jméno,"a"),:);
boys  = T(~endsWith(T.Jméno,"a"),:);
diffHeight = mean(boys.Výška) - mean(girls.Výška);
```

Proč: Rozdělení datasetu na dvě skupiny a porovnání jejich průměrů ukazuje rozdíl mezi pohlavími.

19. **Poměr studentů nad mediánem výšky**

```
mH = median(T.Výška);
ratio = sum(T.Výška > mH) / height(T);
```

Proč: Ukazuje, jak zjistit relativní četnost (proporci) pomocí logické podmínky a funkce `sum`.

4.3.3 Shrnutí

Ukázali jsme, jak základní statistické funkce (`mean`, `median`, `std`, `max`, `min`) fungují nejen na celé tabulce, ale i na vyfiltrovaných podmnožinách. To je základní stavební kámen pro seriózní analýzu dat.

4.4 Čtvrtek – Vizualizace statistik, export výsledků, korelace a lineární regrese

V tomto bloku najdeš 20 úloh (od základních grafů po korelační analýzu a lineární regresi). Používáme stejný dataset `students.csv` (25 studentů). Každá úloha obsahuje: kompaktní MATLAB kód a podrobné, krok-za-krokem vysvětlení **PROČ**.

4.4.1 Poznámka

V některých úlohách používáme funkce, které jsme si ještě explicitně neprobírali (např. `exportgraphics` — použité již dříve ve 4. týdnu). Pokud bys neměl některé toolboxy, většinu úloh lze provést pomocí základních funkcí MATLABu (např. místo `exportgraphics` lze použít `print` nebo `saveas`). Vždy to zmíním.

4.4.2 Úlohy (20)

1. **Histogram rozložení věku (výchozí nastavení).**

```
T = readtable('students.csv','TextType','string');
figure;
histogram(T.Věk);
xlabel('Věk'); ylabel('Počet studentů');
title('Histogram: Rozložení věku studentů');
grid on;
```

Proč:

- `histogram` je základní nástroj pro zobrazení rozložení jedné číselné proměnné.
- Výchozí počet košů (bins) MATLABu je často rozumný, ale v dalších úlohách ukážeme, jak ho upravit.
- Graf dává rychlý přehled, kde je „hustota“ věku.

2. **Histogram váhy s 10 koši a normovaným PDF (hustota pravděpodobnosti).**

```
figure;
histogram(T.Váha, 10, 'Normalization', 'pdf'); % PDF = pravdě
    podobnostní hustota
xlabel('Váha [kg]'); ylabel('Hustota');
title('Histogram váhy (10 košů, normalizovaný na PDF)');
grid on;
```

Proč:

- `'Normalization','pdf'` škáluje histogram tak, aby plocha byla 1 — vhodné pro porovnání s teoretickou hustotou nebo mezi soubory různých velikostí.
- Volba počtu košů (10) ovlivní hladkost rozdělení — méně košů → více vyhlazení.

3. **Kumulativní histogram (Empirická distribuční funkce).**

```
figure;
histogram(T.Věk,'Normalization','cdf'); % cdf = kumulativní
   distribuční funkce
xlabel('Věk'); ylabel('CDF');
title('Kumulativní histogram věku (ECDF aproximace)');
grid on;
```

Proč:

- `'Normalization','cdf'` vykreslí kumulativní podíl — snadno zjistíš medián (kde CDF=0.5) nebo kvantily.
- ECDF (empirická distribuční funkce) je základní deskriptivní nástroj.

4. **Boxplot výšky (box-and-whisker) — identifikace outlierů.**

```
figure;
boxplot(T.Výška);
ylabel('Výška [cm]');
title('Boxplot výšky studentů');
grid on;
```

Proč:

- Boxplot ukazuje medián, kvartily a potenciální odlehlé hodnoty (outliery).
- Je to kompaktní nástroj pro srovnání rozložení mezi skupinami (v příští úloze ukážeme grouped boxplot).

5. **Grouped boxplot: výška podle známky (1..5).**

```
figure;
boxplot(T.Výška, T.Známka);
xlabel('Známka'); ylabel('Výška [cm]');
title('Boxplot: Výška podle známky');
grid on;
```

Proč:

- `boxplot(X,group)` vykreslí separátní boxploty pro každou kategorii (zde `Známka`).
- Usnadňuje vizuální kontrolu, zda např. lepší studenti mají jinou distribuci výšky.

6. **Sloupcový graf (bar): počet studentů v každé známce.**

```
counts = countcats(categorical(T.Známka));
grades = categories(categorical(T.Známka));
figure; bar(str2double(grades), counts);
xlabel('Známka'); ylabel('Počet studentů');
title('Počet studentů podle známky');
grid on;
```

Proč:

- `countcats` spočítá frekvence kategorií; `bar` je vhodný pro diskretizované počty.
- Sloupcový graf je intuitivní pro publikum a dobrý do prezentací.

7. **Histogram výšky a zároveň vykreslení průměru a mediánu jako vertikálních čar.**

```
figure; histogram(T.Výška,12); hold on;
xline(mean(T.Výška),'r--','LineWidth',2,'Label','Mean');
xline(median(T.Výška),'g-','LineWidth',2,'Label','Median');
hold off; xlabel('Výška [cm]'); ylabel('Počet'); title('Výška
    : mean + median');
grid on;
```

Proč:

- Přidáním `xline` jasně vizualizujeme vztah mezi středními hodnotami a rozložením.
- Pomáhá to interpretovat asymetrii dat (skewness): pokud mean $\neq$ median, distribuce je nesymetrická.

8. **Boxplot váhy s označením outlierů a export grafu do PNG (300 DPI).**

```
f = figure('Color','w');
boxplot(T.Váha);
title('Boxplot váhy studentů');
exportgraphics(f,'boxplot_vaha.png','Resolution',300);
```

Proč:

- Export obrázku s 300 DPI je standard pro prezentace/tisk.
- `exportgraphics` zachová kvalitu a umožní vektorový export do PDF, pokud je graf vektorový.

9. **Histogramy výšky rozdělené podle pohlaví (položka: jména končící na 'a' jako dívky).**

```
female = endsWith(T.Jméno,'a');
figure; hold on;
histogram(T.Výška(female),'Normalization','pdf','DisplayName'
    ,'dívky');
histogram(T.Výška(~female),'Normalization','pdf','DisplayName
    ','chlapci');
legend; hold off;
xlabel('Výška [cm]'); ylabel('Hustota'); title('Výška: dívky
    vs. chlapci');
grid on;
```

Proč:

- Porovnání normalizovaných histogramů ukazuje rozdíly tvaru distribucí mezi skupinami bez ohledu na jejich velikost.
- `DisplayName` a `legend` zlepšují čitelnost.

10. **Boxplot BMI (přepočítaného) pro každou známku a export datních statistik do tabulky.**

```
% Přepočet BMI pokud není
height_m = T.Výška/100;
if ~ismember('BMI', T.Properties.VariableNames)
    T.BMI = T.Váha ./ (height_m.^2);
end

% Boxplot BMI podle známky
figure; boxplot(T.BMI, T.Známka);
xlabel('Známka'); ylabel('BMI'); title('BMI podle známky');

% Export základních statistik BMI do CSV
BMI_stats = table(categories(categorical(T.Známka))', ...
    grpstats(T.BMI, T.Známka, {'mean','median','std'}), ...
    'VariableNames', {'Grade', 'Stats'});
writetable(BMI_stats, 'BMI_stats_by_grade.csv');
```

Proč:

- Přidali jsme sloupec BMI (pokud tam není) a porovnali rozložení BMI podle akademického výkonu.
- `grpstats` (Statistics Toolbox) poskytuje rychlou agregaci; pokud není dostupný, lze použít `groupsummary` nebo `splitapply`.
- Ukládáme výsledky do CSV pro další reporting.

11. **Scatter plot výšky vs. váhy s barevnou škálou podle věku.**

```
figure;
scatter(T.Výška, T.Váha, 60, T.Věk, 'filled');
xlabel('Výška [cm]'); ylabel('Váha [kg]');
title('Váha vs. výška (barva = věk)');
colorbar; grid on;
```

Proč:

- `scatter(x,y,s,c)` umožňuje vizualizovat 3. dimenzi pomocí barvy.
- Pomůže odhalit věkové vzory (např. starší studenti těžší/vyšší).

12. **Korelační matice pro číselné proměnné (věk, výška, váha, BMI, známka).**

```
X = [T.Věk, T.Výška, T.Váha, T.BMI, T.Známka];
R = corrcoef(X, 'Rows','pairwise'); % Pearsonův r (pairwise
   ignore NaN)
disp(R);
```

Proč (matematické vysvětlení):

- **Pearsonův korelační koeficient** mezi dvěma proměnnými x a y je definován jako

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}.$$

- `corrcoef` v MATLABu vrací matici korelací mezi sloupci matice X. Volba `'Rows','pairwise'` zajistí využití párových hodnot při případných NaN.
- Hodnota r je mezi -1 a 1; $|r|$ blízko 1 znamená silnou lineární korelaci, 0 znamená žádnou lineární závislost.

13. **Vypočti a vyznač v scatteru korelační koeficient mezi výškou a váhou (ručně i pomocí `corr`).**

```
x = T.Výška; y = T.Váha;
% ruční výpočet Pearson r (bez NaN)
valid = ~isnan(x) & ~isnan(y);
x = x(valid); y = y(valid);
r_manual = sum((x-mean(x)).*(y-mean(y))) / sqrt(sum((x-mean(x
    )).^2)*sum((y-mean(y)).^2));
r_builtin = corr(x,y);
figure; scatter(x,y,'filled'); grid on;
xlabel('Výška'); ylabel('Váha'); title(sprintf('r = %.3f (
    manual), %.3f (corr)', r_manual, r_builtin));
```

Proč:

- Ruční výpočet ilustruje přesnou formuli a numerické kroky.
- `corr` je pohodlné API; porovnání ověřuje správnost.

14. **Jednoduchá lineární regrese: váha jako funkce výšky pomocí `polyfit` (1. stupeň).**

```
x = T.Výška; y = T.Váha;
valid = ~isnan(x) & ~isnan(y);
p = polyfit(x(valid), y(valid), 1); % p(1)=slope, p(2)=
    intercept
slope = p(1); intercept = p(2);
% vykreslení s regresní přímkou
xx = linspace(min(x), max(x), 100);
yy = polyval(p, xx);
figure; scatter(x,y,'filled'); hold on;
plot(xx, yy, 'r-', 'LineWidth',2); hold off;
xlabel('Výška'); ylabel('Váha'); title(sprintf('Regrese: y =
    %.2f x + %.2f', slope, intercept));
grid on;
```

Proč (matematicky):

- Lineární regrese v nejjednodušším smyslu hledá parametry a, b v modelu $y = a + bx$ minimalizující sumu čtverců reziduí (OLS).
- Pro jednorozměrný případ lze ukázat, že

$$b = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}, \qquad a = \bar{y} - b\bar{x}.$$

- `polyfit` používá numericky stabilní metodu pro výpočet koeficientů (nízké pořadí polynomu).

15. **Spočti R-squared (R^2) pro regresi z předešlé úlohy a vysvětli význam.**

```
% předpokládáme p a yy z předchozího kroku
y_hat = polyval(p, x(valid));
SS_res = sum((y(valid)-y_hat).^2);       % suma čtverců reziduí
SS_tot = sum((y(valid)-mean(y(valid))).^2); % celková suma čtverců
R2 = 1 - SS_res/SS_tot;
disp(['R^2 = ', num2str(R2)]);
```

Proč (matematicky):

- $R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$, kde SSE = součet čtverců reziduí, SST = celková suma čtverců.
- R^2 udává, jakou část variance v y vysvětluje model; $R^2 = 1$ znamená dokonalé vysvětlení, $R^2 = 0$ znamená, že model nevysvětluje více než průměr.

16. **Zobrazení regresní přímky s 95% intervalem spolehlivosti (approx.) — jednoduchý přístup pomocí bootstrapu.**

```
% Bootstrap aproximace intervalu predikce (jednoduchá implementace)
nboot = 1000;
b_boot = zeros(nboot,2);
X = x(valid); Y = y(valid);
N = numel(X);
for i=1:nboot
    idx = randsample(N, N, true); % s náhradou
    pb = polyfit(X(idx), Y(idx), 1);
    b_boot(i,:) = pb;
end
% 95% intervaly pro parametry
b_ci = quantile(b_boot, [0.025 0.975]);
% Intervaly predikce pro body xx
yy_boot = zeros(nboot, numel(xx));
for i=1:nboot
    yy_boot(i,:) = polyval(b_boot(i,:), xx);
end
yy_lo = quantile(yy_boot, 0.025);
yy_hi = quantile(yy_boot, 0.975);

figure; scatter(X,Y,'filled'); hold on;
plot(xx, yy, 'r-', 'LineWidth',2);
plot(xx, yy_lo, 'r--'); plot(xx, yy_hi, 'r--');
hold off; title('Regrese s bootstrap 95% intervalem');
xlabel('Výška'); ylabel('Váha'); grid on;
```

Proč:

- Přesný analytický interval vyžaduje předpoklady o šumu (normalita, konstantní variance). Bootstrap je robustní numerická metoda pro aproximaci nejistot.

- Zde provádíme jednoduchý bootstrap parametrů a následně promítáme rozptyl do predikcí.

17. **Korelace (a scatter + regrese) mezi věkem a známkou; interpretuj směr a význam korelace.**

```
x_age = T.Věk; y_grade = T.Známka;
valid = ~isnan(x_age) & ~isnan(y_grade);
r_age_grade = corr(x_age(valid), y_grade(valid));
figure; scatter(x_age, y_grade, 'filled'); grid on;
xlabel('Věk'); ylabel('Známka'); title(sprintf('r(age,grade)
   = %.3f', r_age_grade));
% lineární fit pro vizualizaci
p_ag = polyfit(x_age(valid), y_grade(valid), 1);
xx_ag = linspace(min(x_age), max(x_age), 100);
plot(xx_ag, polyval(p_ag, xx_ag), 'r-', 'LineWidth', 1.5);
```

Proč:

- Korelace mezi věkem a známkou může naznačit, zda starší studenti mají lepší/horší výsledky (kladná korelace: s věkem roste známka — pozor: v ČR menší číslo je lepší známka; interpretace značí opačný efekt).
- Regresní přímka slouží k vizuální pomoci — nesmí se zaměňovat s kauzalitou.

18. **Ulož všechny hlavní výsledky analýzy (korelační matici, parametry regrese, R2) do MAT souboru a CSV.**

```
results.R = R;       % korelace
results.p = p;       % koeficienty výšky->váha (z úlohy 14)
results.R2 = R2;
save('analysis_results.mat','results');

% Tabulka výsledků (jednoduchý přehled) a export CSV
ResTable = table({'height_weight'; 'age_grade'}, ...
    [p(1); p_ag(1)], [p(2); p_ag(2)], [R2; NaN], '
       VariableNames', {'Model','Slope','Intercept','R2'});
writetable(ResTable,'regression_summary.csv');
```

Proč:

- `save` uloží MATLAB proměnné v binárním formátu `.mat` (rychlé načtení později).
- `writetable` vytváří CSV soubor, vhodný pro sdílení nebo dokumentaci.

19. **Heatmap korelační matice (visualizace síly korelace).**

```
figure;
imagesc(R); colorbar; axis equal tight;
xticks(1:5); yticks(1:5);
xticklabels({'Age','Height','Weight','BMI','Grade'});
yticklabels({'Age','Height','Weight','BMI','Grade'});
title('Heatmap korelační matice (Pearson r)');
```

Proč:

- Heatmapa je intuitivní způsob, jak rychle najít silné/ slabé korelace (barevné škály).
- `imagesc` mapuje hodnoty na barvy; `colorbar` ukazuje legendu.

20. **Srovnání dvou modelů: lineární regrese vs. nulová model (průměr) — porovnej SSE a vizualizuj rezidua.**

```
% Předpokládáme model y ~ weight predicted from height (p)
y = T.Váha; x = T.Výška;
valid = ~isnan(x) & ~isnan(y);
y = y(valid); x = x(valid);
y_hat = polyval(p, x);
SSE_model = sum((y - y_hat).^2);
SSE_null = sum((y - mean(y)).^2); % nulový model = konstantní
    průměr
disp(['SSE model: ', num2str(SSE_model)]);
disp(['SSE null : ', num2str(SSE_null)]);

% Rezidua a jejich histogram
resid = y - y_hat;
figure;
subplot(1,2,1); scatter(x, resid,'filled'); xlabel('Výška');
   ylabel('Rezidua'); title('Rezidua vs. výška');
subplot(1,2,2); histogram(resid); xlabel('Rezidua'); title('
   Histogram reziduí');
```

Proč:

- Porovnání SSE (suma čtverců reziduí) dává přímé měřítko, zda model lépe vysvětluje data než jednoduchý průměr.
- Analýza reziduí (graf vs. prediktoru a histogram) pomáhá diagnostikovat porušení předpokladů (heteroskedasticita, nelinearita, ne-normálnost).

4.4.3 Doplňující poznámky, interpretace a best practices

- **Interpretace korelace:** Pearsonův r měří lineární vztah. $|r| > 0.7$ se obvykle považuje za silnou korelaci, 0.3–0.7 střední, <0.3 slabou — ale rozhodnutí závisí na kontextu.
- **Kauzalita vs. korelace:** korelace neimplikuje kauzalitu. Pozor na zkreslení (confoundery).
- **Regrese:** před použitím lineární regrese ověř předpoklady (linearity, normalita reziduí, homogenita variance). Pokud nejsou splněny, zvaž transformace nebo nelineární modely.
- **Export grafů:** pro vědecké články preferuj vektorové formáty (PDF) — `exportgraphics(f,'plot` Pro prezentace PNG s 300 DPI.
- **Robustnost:** Pokud jsou v datech outliery, zváž robustní regresi (např. `robustfit`) — to je rozšíření mimo základní příkazy, ale užitečné v praxi.

4.5 Pátek – Základy práce s `datetime`

V této části se naučíme pracovat s časovými daty v MATLABu. Budeme používat vestavěný typ `datetime`, který umožňuje pohodlně ukládat datum a čas, provádět výpočty, formátovat výstupy a kombinovat je s datovými tabulkami (`table`, `timetable`).

Stejně jako v předchozích dnech připravíme 20 úloh, ke každé ukázkový MATLAB kód a podrobné vysvětlení **PROČ**.

—

4.5.1 Úlohy (20)

1. **Vytvoření aktuálního času pomocí `datetime('now')`.**

```
t = datetime('now')
```

Proč: `datetime` ukládá datum a čas s vysokou přesností. Volba ''now'' znamená aktuální okamžik podle systémového času.

2. **Vytvoření konkrétního data.**

```
t = datetime(2025,9,8,14,30,0) % 8. září 2025, 14:30:00
```

Proč: Lze zadat rok, měsíc, den, hodinu, minutu, sekundu. Hodí se pro definici testovacích časů.

3. **Vytvoření vektoru časů (např. po dnech).**

```
d = datetime(2025,1,1) : days(1) : datetime(2025,1,10)
```

Proč: Operátory typu `days(1)`, `hours(1)` definují časový krok. Toto je analogické klasickému `start:step:end` u čísel.

4. **Rozdíl dvou časů.**

```
t1 = datetime(2025,1,1);
t2 = datetime(2025,2,1);
dt = t2 - t1
```

Proč: Výsledkem je objekt `calendarDuration`, který uchovává rozdíl v měsících/dnech.

5. **Výpočet rozdílu ve dnech (číslo).**

```
days_between = days(t2 - t1)
```

Proč: Převádíme rozdíl na čisté číslo (počet dní). Vhodné pro analýzy.

6. **Převod textu na datetime.**

```
t = datetime('08-Sep-2025 14:30:00','InputFormat','dd-MMM-
    yyyy HH:mm:ss')
```

Proč: `datetime` umí číst textová data, ale musíme zadat správný formát.

7. **Změna výstupního formátu.**

```
t = datetime('now');
t.Format = 'dd.MM.yyyy HH:mm'
```

Proč: Formát výstupu lze měnit podle potřeby (pro tabulky, grafy, reporting).

8. **Výběr části data (rok, měsíc, den).**

```
t = datetime('now');
year(t), month(t), day(t)
```

Proč: Rychle extrahujeme složky data pro filtraci nebo statistiku.

9. **Výběr času (hodina, minuta, sekunda).**

```
hour(t), minute(t), second(t)
```

Proč: Stejný princip pro časové složky.

10. **Přičítání časových jednotek.**

```
t = datetime('2025-09-08 14:30:00');
t_plus = t + hours(5) + minutes(30)
```

Proč: Jednoduše lze posouvat čas o definované intervaly.

11. **Rozsah po minutách.**

```
d = datetime(2025,9,8,0,0,0) : minutes(30) : datetime
   (2025,9,8,6,0,0)
```

Proč: Ideální pro simulace měření (např. teplota každých 30 minut).

12. **Kontrola dne v týdnu.**

```
d = datetime(2025,9,8);
day_of_week = day(d,'name')
```

Proč: Funkce `day` s volbou ''name'' dává textový název dne (pondělí, úterý, ...).

13. **Kombinace s tabulkou dat (čas jako index).**

```
time = datetime(2025,9,1) + days(0:4);
values = rand(5,1);
T = table(time', values, 'VariableNames',{'Čas','Měření'})
```

Proč: Čas lze přímo uložit jako sloupec tabulky, což usnadňuje grafy a analýzu.

14. **Převod tabulky na časovou tabulku (`timetable`).**

```
TT = table2timetable(T,'RowTimes','Čas')
```

Proč: `timetable` umožňuje operace podle času (resampling, synchronizace, filtrování).

15. **Resampling časových dat (průměry po dni).**

```
TT_daily = retime(TT,'daily','mean')
```

Proč: `retime` umožňuje převést měření s vyšší frekvencí na agregovaná data (např. denní průměry).

16. **Výpočet rozdílů mezi po sobě jdoucími časy.**

```
dt = diff(TT.Čas)
minutes(dt)
```

Proč: Získáme intervaly mezi měřeními. Lze zjistit, zda jsou rovnoměrné.

17. **Logické filtrování podle data.**

```
sel = T.Čas > datetime(2025,9,2) & T.Čas < datetime(2025,9,4)
    ;
T(sel,:)
```

Proč: Stejně jako u čísel lze filtrovat řádky tabulky podle časového rozsahu.

18. **Graf hodnot proti času.**

```
plot(T.Čas, T.Měření,'-o');
xlabel('Čas'); ylabel('Měření');
title('Měření v čase');
grid on;
```

Proč: Pokud osa X obsahuje `datetime`, MATLAB ji automaticky naformátuje.

19. **Převod časové zóny.**

```
t = datetime('now','TimeZone','UTC');
t_local = datetime(t,'TimeZone','Europe/Prague')
```

Proč: Práce s časovými pásmy je zásadní při zpracování dat ze senzorů a serverů.

20. **Zaokrouhlování času (na minuty, hodiny).**

```
t = datetime('now');
round(t,'minute')
floor(t,'hour')
```

Proč: Užitečné pro synchronizaci měření na pravidelný rastr (např. každých 5 min).

—

4.5.2 Shrnutí

- **datetime** je základní datový typ pro práci s časem v MATLABu.
- Umí reprezentovat jednotlivé okamžiky, vektory časů i celé časové osy.
- Snadno se kombinuje s tabulkami a s grafickými funkcemi.
- Umí převody mezi textem, časovými zónami a různé formátování.
- V praxi je velmi výhodný pro práci s experimentálními daty měřenými v čase.

4.6 Sobota – Projekt: Analýza datasetu spotřeby energie + pokročilé operace s časem

Tento blok shrnuje **větší propojený projekt**, který kombinuje všechny dosavadní znalosti:

- Načítání dat (CSV, tabulky s časem a hodnotami).
- Práce s `datetime`, filtrování a manipulace s časem.
- Vizualizace trendů (časové řady, histogramy, boxploty).
- Pokročilé operace s časem: časová pásma, resampling, pohyblivé průměry, výpočty rozdílů v čase.
- Export výsledků.

Ukázkový dataset (`energy.csv`) bude obsahovat 30 záznamů: čas (každou hodinu) a spotřebu energie (v kWh). Data simulujeme ručně, aby byla realistická.

—

4.6.1 Ukázkový dataset (energy.csv)

```
Čas,Spotreba_kWh
2025-09-01 00:00:00,2.1
2025-09-01 01:00:00,1.8
2025-09-01 02:00:00,1.6
2025-09-01 03:00:00,1.5
2025-09-01 04:00:00,1.4
2025-09-01 05:00:00,1.6
2025-09-01 06:00:00,2.0
2025-09-01 07:00:00,2.5
2025-09-01 08:00:00,3.2
2025-09-01 09:00:00,3.8
2025-09-01 10:00:00,4.1
2025-09-01 11:00:00,4.5
2025-09-01 12:00:00,4.9
2025-09-01 13:00:00,5.2
2025-09-01 14:00:00,5.0
2025-09-01 15:00:00,4.8
2025-09-01 16:00:00,4.4
2025-09-01 17:00:00,4.1
2025-09-01 18:00:00,4.8
2025-09-01 19:00:00,5.5
2025-09-01 20:00:00,5.9
2025-09-01 21:00:00,5.2
2025-09-01 22:00:00,4.3
2025-09-01 23:00:00,3.2
2025-09-02 00:00:00,2.5
2025-09-02 01:00:00,2.0
2025-09-02 02:00:00,1.7
```

```
2025-09-02 03:00:00,1.5
2025-09-02 04:00:00,1.4
2025-09-02 05:00:00,1.6
```

—

4.6.2 Krok 1 – Načtení dat a převod na časovou tabulku

```
% Načtení CSV souboru
T = readtable('energy.csv');
 T.Jm_no


% Pro zachování původních názvů sloupců
T = readtable("energy.csv", 'VariableNamingRule','preserve');
T.("Čas")   % musíte použít uvozovky !!!!!


% Převedeme sloupec 'Čas' na datetime
T.Čas = datetime(T.Čas,'InputFormat','yyyy-MM-dd HH:mm:ss');

% Převod na timetable
TT = table2timetable(T,'RowTimes','Čas');
```

Proč: Tabulka `timetable` je ideální pro práci s časovou osou – MATLAB chápe, že indexem řádků je čas, a umožňuje resampling nebo filtrování podle časového rozsahu.

—

4.6.3 Krok 2 – Vizualizace trendu spotřeby

```
plot(TT."Čas", TT.Spotreba_kWh,'-o');
xlabel('Čas'); ylabel('Spotřeba [kWh]');
title('Denní profil spotřeby energie');
grid on;
```

Proč: Spotřeba má jasný denní cyklus (ráno a večer maxima, v noci minima). Vizualizace odhaluje trend lépe než samotná čísla.

—

4.6.4 Krok 3 – Statistické zpracování

```
prumer = mean(TT.Spotreba_kWh);
median_val = median(TT.Spotreba_kWh);
odchylka = std(TT.Spotreba_kWh);
```

Proč: Získáme základní charakteristiky datasetu – průměrná spotřeba, rozptyl, robustnější medián.

—

4.6.5 Krok 4 – Histogram a boxplot

```
figure;
subplot(1,2,1);
histogram(TT.Spotreba_kWh);
xlabel('Spotřeba [kWh]'); ylabel('Četnost');
title('Histogram spotřeby');

subplot(1,2,2);
boxplot(TT.Spotreba_kWh);
ylabel('Spotřeba [kWh]');
title('Boxplot spotřeby');
```

Proč: Histogram ukazuje rozložení hodnot, boxplot zvýrazní medián, kvartily a extrémy.

—

4.6.6 Krok 5 – Pokročilé operace s časem

1. **Resampling (po 6 hodinách, průměry):**

```
TT6h = retime(TT,'regular','mean','TimeStep',hours(6));
```

Proč: Agregujeme spotřebu na větší časový krok – vyhlazení trendu.

2. **Pohyblivý průměr (3-hodinové okno):**

```
TT.MA3 = movmean(TT.Spotreba_kWh,3);
```

Proč: Pohybující se průměr odfiltruje krátkodobé výkyvy.

3. **Rozdíly mezi měřeními:**

```
delta = diff(TT.Spotreba_kWh);
```

Proč: Získáme přírůstky/úbytky spotřeby mezi hodinami.

4. **Časová pásma:**

```
TT_Time_TimeZone = 'UTC';
TT_local = datetime(TT."Čas",'TimeZone','Europe/Prague');
```

Proč: Pro reálná data (např. smart metering) je práce s časovými pásmy nutností.

—

4.6.7 Krok 6 – Vizualizace pohyblivého průměru

```
plot(TT.'Čas', TT.Spotreba_kWh,'-o'); hold on;
plot(TT.'Čas', TT.MA3,'-r','LineWidth',2);
xlabel('Čas'); ylabel('Spotřeba [kWh]');
title('Spotřeba vs. pohyblivý průměr');
legend('Měření','MA (3 h)');
grid on;
```

Proč: Na grafu vidíme, jak pohyblivý průměr vyhlazuje „špičky“ a ukazuje dlouhodobější trend.

—

4.6.8 Krok 7 – Export výsledků

```
% Uložení resamplovaných dat
writetable(timetable2table(TT6h),'energy_6h.csv');

% Export grafu ve vysokém rozlišení
exportgraphics(gcf,'energy_trend.png','Resolution',300);
```

Proč: Umožní využít výsledky v prezentacích, zprávách nebo pro další zpracování.

—

4.6.9 Úlohy k projektu

1. Načtěte dataset a ověřte, zda jsou časy správně převedeny na `datetime`.
2. Vytvořte graf spotřeby a popište, kde jsou maxima a minima.
3. Spočítejte průměrnou, maximální a minimální spotřebu.
4. Určete hodinu, kdy byla nejvyšší spotřeba.
5. Najděte období, kdy spotřeba byla menší než 2 kWh.
6. Vykreslete histogram spotřeby.
7. Vytvořte boxplot a okomentujte rozptyl.
8. Proveďte resampling po 3 hodinách a porovnejte s původními daty.
9. Spočítejte pohyblivý průměr s oknem 5.
10. Vykreslete graf s pohyblivým průměrem a okomentujte rozdíl.
11. Určete rozdíly mezi po sobě jdoucími měřeními.
12. Najděte okamžik s největším nárůstem spotřeby.
13. Přepněte časové pásmo na UTC a zobrazte data.
14. Přepněte časové pásmo zpět na Prahu.
15. Zaokrouhlete časy na celé hodiny a porovnejte s původními.
16. Vytvořte korelační graf spotřeby mezi dvěma dny.
17. Uložte denní průměry do nového CSV.
18. Exportujte graf do PNG a PDF.
19. Okomentujte, zda má dataset pravidelný časový krok.
20. Navrhněte, jak by se dala spotřeba predikovat pomocí lineární regrese.

—

4.6.10 Shrnutí projektu

- Dataset spotřeby energie jsme načetli, převedli na časovou tabulku a analyzovali.
- Vykreslili jsme trend, základní statistiky a vizualizace (histogram, boxplot).
- Provedli jsme pokročilé operace s časem: resampling, pohyblivé průměry, rozdíly v čase, časová pásma.
- Výsledky jsme exportovali pro další použití.

4.6.11 Detailní řešení úloh (1–20) — MATLAB kód + podrobná vysvětlení

Níže jsou pro každou z 20 úloh kompletní příkazy, které můžeš zkopírovat do MATLABu, a důkladné vysvětlení **proč** jsou tyto kroky použity a na co si dát pozor.

1. **Načtěte dataset a ověřte, zda jsou časy správně převedeny na `datetime`.**

```
% 1) Načtení CSV a kontrola datetime
T = readtable('energy.csv','TextType','string');    % načtení surového CSV
% převod sloupce (nazvaného 'Čas') na datetime s explicitním formátem
T.('Čas') = datetime(T.('Čas'),'InputFormat','yyyy-MM-dd HH:mm:ss','TimeZone','');
% zkontrolujeme typ a prvních pár hodnot
whos T
disp(T(1:6,:));
% převod na timetable pro další kroky
TT = table2timetable(T,'RowTimes','Čas');
```

Proč:

- `readtable` vrací `table` se sloupci podle hlavičky; parametr `'TextType','string'` je vhodný pokud jsou v CSV texty s diakritikou.
- Explicitní `datetime(...,'InputFormat',...)` zajistí jednoznačné parsování. Bez něho MATLAB někdy hádá formát a může nastat špatné rozpoznání (zvláště při lokálních nastaveních).
- `whos` a `disp` slouží k ověření, že sloupec je nyní typu `datetime` a že hodnoty dávají smysl.
- `table2timetable` vytvoří `timetable`, což je doporučený typ pro časové řady v MATLABu.

2. **Vytvořte graf spotřeby a popište, kde jsou maxima a minima.**

```
% 2) Graf časové řady a označení maxima/minima
figure('Color','w');
plot(TT.('Čas'), TT.Spotreba_kWh, '-o','LineWidth',1.2,'MarkerSize',4);
xlabel('Čas'); ylabel('Spotřeba [kWh]');
title('Spotřeba energie (hodinová)');
grid on;

% nalezení maxima a minima
[peakVal, peakIdx] = max(TT.Spotreba_kWh);
[lowVal, lowIdx]   = min(TT.Spotreba_kWh);
% označíme je v grafu
hold on;
plot(TT.('Čas')(peakIdx), peakVal, 'r*','MarkerSize',10);
text(TT.('Čas')(peakIdx), peakVal, sprintf('  max = %.2f kWh',peakVal),'Color','r');
```

```
plot(TT.('Čas')(lowIdx), lowVal, 'b*','MarkerSize',10);
text(TT.('Čas')(lowIdx), lowVal, sprintf('  min = %.2f kWh',
   lowVal),'Color','b');
hold off;
```

Proč:

- Základní vizualizace odhalí denní cyklus — ranní a večerní špičky, noční poklesy.
- `max` a `min` vrátí hodnoty a indexy; indexy umožní najít čas, kdy k těmto extrémům došlo.
- Přidání značek a textu zlepšuje čitelnost výsledného grafu (užitečné do reportu).

3. **Spočítejte průměrnou, maximální a minimální spotřebu.**

```
% 3) Základní statistiky
mean_consumption = mean(TT.Spotreba_kWh,'omitnan');
max_consumption  = max(TT.Spotreba_kWh);
min_consumption  = min(TT.Spotreba_kWh);

fprintf('Průměr: %.3f kWh, Max: %.3f kWh, Min: %.3f kWh\n',
   ...
    mean_consumption, max_consumption, min_consumption);
```

Proč:

- `mean(...,'omitnan')` je bezpečné i když jsou v datech NaN — ignoruje je.
- Tyto tři metriky dávají rychlý numerický přehled o rozsahu a centrální hodnotě spotřeby.

4. **Určete hodinu, kdy byla nejvyšší spotřeba.**

```
% 4) Hodina s nejvyšší spotřebou
[peakVal, peakIdx] = max(TT.Spotreba_kWh);
peakTime = TT.('Čas')(peakIdx);
fprintf('Nejvyšší spotřeba %.2f kWh byla v %s\n', peakVal,
   datestr(peakTime,'yyyy-mm-dd HH:MM'));
```

Proč:

- Index vrácený `max` lze použít přímo pro výběr časové značky z `timetable`.
- Formátování pomocí `datestr` nebo `char` poskytne čitelný řetězec - vhodné do zprávy.

5. **Najděte období, kdy spotřeba byla menší než 2 kWh.**

```
% 5) Intervaly kdy spotřeba < 2 kWh
mask_low = TT.Spotreba_kWh < 2;
times_low = TT.('Čas')(mask_low);
% Pokud chceme segmenty spojit dohromady:
% najdeme přechody false->true a true->false
dmask = [0; diff(double(mask_low))];
startIdx = find(dmask == 1);
```

```
endIdx   = find(dmask == -1) - 1;
% hraní s okraji: pokud maska začíná True
if mask_low(1); startIdx = [1; startIdx]; end
if mask_low(end); endIdx = [endIdx; length(mask_low)]; end

% výpis intervalů
for k = 1:length(startIdx)
    fprintf('Perioda %d: %s       %s\n', k, datestr(TT.('Čas'
       )(startIdx(k))), datestr(TT.('Čas')(endIdx(k))));
end
```

Proč:

- Jednoduché maskování vrátí hodiny s malou spotřebou; pro lidské čtení je lepší spojit souvislé hodiny do intervalů.
- Metoda s `diff` nachází začátky a konce bloků True; ošetřujeme hraniční případy.

6. **Vykreslete histogram spotřeby.**

```
% 6) Histogram spotřeby s vhodným počtem košů
figure('Color','w');
histogram(TT.Spotreba_kWh, 10); % 10 košů
xlabel('Spotřeba [kWh]'); ylabel('Počet hodin');
title('Histogram spotřeby energie');
grid on;
```

Proč:

- Histogram odhalí tvar rozdělení (jednomodální, vícemodální), existence špiček apod.
- Volba počtu košů je heuristická; 8–12 je často rozumné pro malý dataset.

7. **Vytvořte boxplot a okomentujte rozptyl.**

```
% 7) Boxplot
figure('Color','w');
boxplot(TT.Spotreba_kWh);
ylabel('Spotřeba [kWh]');
title('Boxplot hodinové spotřeby');
grid on;
```

Proč:

- Boxplot zobrazuje medián, Q1, Q3 a potenciální outliery. Sledujeme, zda jsou některé hodiny výrazně odlišné (špičky).
- U malé datové sady interpretuj outlier opatrně — může to být skutečná špička nebo chyba měření.

8. **Proveďte resampling po 3 hodinách a porovnejte s původními daty.**

```
% 8) Retime na 3-hodinové průměry
TT3h = retime(TT,'regular','mean','TimeStep',hours(3));

% srovnání graficky
figure('Color','w');
plot(TT.('Čas'), TT.Spotreba_kWh,'-o','DisplayName','1h');
   hold on;
plot(TT3h.Time, TT3h.Spotreba_kWh,'-s','LineWidth',1.8,'
   DisplayName','3h mean');
hold off; legend('Location','best');
xlabel('Čas'); ylabel('Spotřeba [kWh]'); title('Srovnání 1h
   vs 3h agregace'); grid on;
```

Proč:

- `retime(...,'regular','mean','TimeStep',hours(3))` vytvoří pravidelnou mřížku s 3hodinovým krokem a spočítá průměry v každém kroku.
- Resampling redukuje šum a zvýrazní dlouhodobější trendy.

9. **Spočítejte pohyblivý průměr s oknem 5 (5 hodin).**

```
% 9) Pohyblivý průměr (window=5)
TT.MA5 = movmean(TT.Spotreba_kWh, 5, 'Endpoints','shrink'); %
    'shrink' zkracuje okno na okrajích
% alternativně 'fill' nebo implicitní chování
```

Proč:

- `movmean` aplikuje průměr přes klouzavé okno — potlačí krátkodobé výkyvy.
- Volba parametru `'Endpoints'` ovlivní chování na okrajích (kde není plné okno). `'shrink'` použije menší okno u okrajů.

10. **Vykreslete graf s pohyblivým průměrem a okomentujte rozdíl.**

```
% 10) Vizualizace s MA5
figure('Color','w');
plot(TT.('Čas'), TT.Spotreba_kWh,'-o','DisplayName','Originá
   lní 1h', 'MarkerSize',4); hold on;
plot(TT.('Čas'), TT.MA5,'-r','LineWidth',2,'DisplayName','MA
   (5)');
xlabel('Čas'); ylabel('Spotřeba [kWh]');
title('Spotřeba + pohyblivý průměr (5 hodin)');
legend('Location','best'); grid on; hold off;
```

Proč:

- MA vyhlazuje krátkodobé špičky; na grafu uvidíš, že maximální hodnoty MA jsou nižší než okamžité špičky, ale lépe reflektují trend.

11. **Určete rozdíly mezi po sobě jdoucími měřeními.**

```
% 11) Rozdíly mezi po sobě jdoucími měřeními
delta = diff(TT.Spotreba_kWh);     % délka N-1
time_delta = TT.('Čas')(2:end);       % časové značky pro
   delta
% zobrazíme první hodnoty
table(time_delta(1:6), delta(1:6), 'VariableNames', {'Time','
   Delta_kWh'})
```

Proč:

- `diff` dává okamžité změny (přírůstky) mezi měřeními — užitečné pro identifikaci náhlých nárůstů nebo poklesů.
- Protože `diff` produkuje kratší vektor, časové značky posuneme o jeden prvek.

12. **Najděte okamžik s největším nárůstem spotřeby.**

```
% 12) Největší nárůst mezi po sobě jdoucími hodinami
[largestInc, idxInc] = max(delta);
time_of_inc = time_delta(idxInc);
fprintf('Největší nárůst %.2f kWh v intervalu začínajícím %s\
   n', largestInc, datestr(time_of_inc,'yyyy-mm-dd HH:MM'));
```

Proč:

- Hledáme maximum z vektoru rozdílů; jeho index přeložíme na čas začátku intervalu, kde k nárůstu došlo.

13. **Přepněte časové pásmo na UTC a zobrazte data.**

```
% 13) Nastavení TimeZone na UTC pro timetable
TT.('Čas')Zone = 'UTC';   % TT.('Čas')Zone nastaví časové pá
   smo všech časových záznamů
% zobrazíme pár řádků
TT(1:6,:)
```

Proč:

- Nastavení `TimeZone` neupraví okamžiky v absolutním smyslu — pouze interpretuje/ukáže je v daném pásmu. To je důležité při srovnání dat z různých zdrojů.

14. **Přepněte časové pásmo zpět na Prahu.**

```
% 14) Zpět do Europe/Prague
TT.('Čas')Zone = 'Europe/Prague';
% kontrola
TT(1:6,:)
```

Proč:

- Přepínání pásma se hodí při vizualizaci pro lokální publikum nebo při porovnávání s UTC-časovými razítky serverů.

- Pokud chceš převést čas absolutně (změnit okamžiky), použij konstrukce s explicitním převodem (např. vytvoření nového datetime s jiným TimeZone).

15. **Zaokrouhlete časy na celé hodiny a porovnejte s původními.**

```
% 15) Zaokrouhlení času na hodiny
roundedTimes = round(TT.('Čas'),'hour');   % zaokrouhlí na
   nejbližší hodinu
% porovnání (pokud bylo původně právě celé hodiny, shodují se
   )
table(TT.('Čas')(1:8), roundedTimes(1:8), 'VariableNames', {'
   Original','Rounded'})
```

Proč:

- Zaokrouhlení je užitečné, pokud pracuješ s daty, kde menší sub-hodinové odchylky nejsou relevantní nebo když chceš agregovat na hodinový rastr.
- Funkce `round` pracuje i s minutami a dalšími jednotkami (např. 'minute','second').

16. **Vytvořte korelační graf spotřeby mezi dvěma dny.**

```
% 16) Korelace mezi spotřebou dvou dní (porovnání hodin)
% extrahujeme data pro 1. a 2. den (předpoklad: data po hodin
   ách)
day1 = TT(TT.Date == datetime(2025,9,1), :);
day2 = TT(TT.Date == datetime(2025,9,2), :);

% zarovnáme podle hodin (pokud chybí hodiny, použijeme join/
   synchronizaci)
% zde přímé porovnání pokud mají stejnou délku a pořadí
if height(day1) == height(day2)
    figure('Color','w');
    scatter(day1.Spotreba_kWh, day2.Spotreba_kWh, 'filled');
    xlabel('Spotřeba 2025-09-01 [kWh]'); ylabel('Spotřeba
       2025-09-02 [kWh]');
    title('Porovnání spotřeby po hodinách mezi dvěma dny');
    grid on;
    r = corr(day1.Spotreba_kWh, day2.Spotreba_kWh,'Rows','
       pairwise');
    fprintf('Pearson r mezi dny = %.3f\n', r);
else
    warning('Dny nemají stejný počet měření     je třeba data
        zarovnat.');
end
```

Proč:

- Porovnání hodinových profilů dvou dní ukáže opakovatelnost chování (podobnost denních profilů).
- Pokud data nejsou zarovnána, je třeba použít `synchronize` nebo jiný join podle času.

17. **Uložte denní průměry do nového CSV.**

```
% 17) Denní průměry a export
TT_daily = retime(TT,'daily','mean');   % denní průměry
daily_table = timetable2table(TT_daily,'RowTimes','Date');
writetable(daily_table,'energy_daily_mean.csv');
```

Proč:

- Denní průměry snižují datovou dimenzi a jsou často vhodné pro reporting nebo modelování s delší dobou kroku.
- `writetable` umožní další zpracování v Excelu nebo jiných nástrojích.

18. **Exportujte graf do PNG a PDF.**

```
% 18) Export grafu: doporučeno mít aktivní figure
f = figure('Color','w');
plot(TT.('Čas'), TT.Spotreba_kWh,'-o'); xlabel('Čas'); ylabel
   ('Spotřeba'); title('Spotřeba');
grid on;
% uložíme do PNG (rastrově) a PDF (vektor)
exportgraphics(f,'energy_plot.png','Resolution',300);
exportgraphics(f,'energy_plot.pdf','ContentType','vector');
```

Proč:

- PNG 300 DPI je vhodné pro prezentace; PDF jako vektor pro publikaci v LaTeXu.
- `exportgraphics` zachová kvalitní texty (LaTeX-like) a je doporučené oproti staršímu `saveas`.

19. **Okomentujte, zda má dataset pravidelný časový krok.**

```
% 19) Kontrola pravidelnosti kroku
dt = minutes(diff(TT.('Čas')));   % v minutách
unique_steps = unique(dt);
disp('Unikátní kroky v minutách:'); disp(unique_steps);
% pokud je jen jeden unikátní krok, dataset je pravidelný
if numel(unique_steps)==1
    fprintf('Dataset má pravidelný časový krok: %d minut\n',
       unique_steps);
else
    fprintf('Dataset nemá zcela pravidelný krok (různé
       hodnoty): '); disp(unique_steps');
end
```

Proč:

- Pro většinu operací (resampling, movmean s fixním oknem) je žádoucí pravidelný krok. Pokud krok není pravidelný, je lepší data nejprve zresamplovat na pravidelnou mřížku nebo použít vážené filtry.

20. **Navrhněte, jak by se dala spotřeba predikovat pomocí lineární regrese.**

```
% 20) Jednoduchý návrh: predikce následující hodiny pomocí předchozích hodnot
% vytvoříme regresní dataset: y_t = Spotreba(t), X = [Spotreba(t-1), Spotreba(t-2), ...]
lag = 3; % použijeme 3 zpožděné hodnoty jako prediktory
Y = TT.Spotreba_kWh((lag+1):end);
N = length(Y);
X = zeros(N, lag);
for k=1:lag
    X(:,k) = TT.Spotreba_kWh((lag+1-k):(end-k));
end
% jednoduchá lineární regrese (OLS) - fit coefficients
beta = (X'*X)\(X'*Y);   % (X'X)^{-1} X' Y
% predikce a RMSE
Yhat = X * beta;
rmse = sqrt(mean((Y - Yhat).^2));
fprintf('Koeficienty (lag1..lag%d): %s; RMSE = %.3f kWh\n', lag, num2str(beta','%.4f '), rmse);
% Pozn.: pro robustní modely doporučit ARIMA, SARIMAX nebo ML přístupy
```

Proč:

- Jednoduchý autoregresivní přístup (použití předchozích hodnot jako prediktorů) je intuitivní a často funkční pro denní/periodické signály.
- Koeficienty se odhadnou OLS; RMSE (root mean squared error) dává měřítko chyby.
- Pokud data obsahují silnou sezónnost (denní profil), je vhodné přidat další proměnné (hodina dne jako dummy, den v týdnu, pohyblivé průměry) nebo použít specializované časové modely (ARIMA, Prophet, LSTM).

4.6.12 Závěrečné doporučení k projektu

- Při experimentování používej malý počet kroků a průběžně kontroluj výsledky (plot, head, summary).
- Pro produkční analýzu: zálohuj surová data, vytvářej skripty, které od začátku do konce pipeline opakují (ETL).
- Pro predikci zvaž robustní validaci: split train/test, křížovou validaci, diagnostiku reziduí.
- V dokumentu/presentaci exportuj grafy jako PDF pro tisk a PNG pro prezentaci.

4.7 Neděle – Načítání a ukládání dat v různých formátech

V této kapitole procvičíme načítání a ukládání dat v různých formátech, které MATLAB běžně používá: CSV, Excel, MAT, textové soubory. Použijeme přitom dataset `students.csv` a vytvoříme také nové soubory.

4.7.1 Úlohy

1. **Načtení CSV pomocí `readtable`**

```
T = readtable('students.csv');
```

Proč: `readtable` načítá CSV do tabulky (`table`), což je vhodné, když máme pojmenované sloupce. Sloupce pak můžeme snadno oslovovat podle názvu, např. `T.Vek`.

2. **Načtení CSV pomocí `readmatrix`**

```
M = readmatrix('students.csv');
```

Proč: `readmatrix` načítá jen číselná data. Textové hlavičky se ztratí. Je to užitečné, pokud nám jde jen o čísla (např. výpočty statistik).

3. **Načtení jen části tabulky**

```
opts = detectImportOptions('students.csv');
opts.SelectedVariableNames = {'Jmeno','Vek'};
T2 = readtable('students.csv', opts);
```

Proč: Můžeme si vybrat jen určité sloupce (např. jména a věk). Šetříme tím paměť a zaměřujeme se jen na data, která potřebujeme.

4. **Uložení tabulky zpět do CSV**

```
writetable(T, 'students_copy.csv');
```

Proč: Funkce `writetable` uloží tabulku do CSV včetně hlavičky. To je ideální pro export a sdílení dat.

5. **Export do Excelu**

```
writetable(T, 'students.xlsx');
```

Proč: Umožňuje předat data uživatelům, kteří pracují v Excelu. MATLAB automaticky vytvoří list s daty.

6. **Čtení z Excelu**

```
T_excel = readtable('students.xlsx');
```

Proč: Stejná funkce `readtable` pracuje i s Excel soubory. Snadno tedy přepínáme mezi formáty.

7. **Uložení jen vybraných dat do Excelu**

```
writetable(T(:,{'Jmeno','Znamka'}), 'grades.xlsx');
```

Proč: Exportujeme jen dvě proměnné – jméno a známku. To se hodí při sdílení výsledků hodnocení.

8. **Uložení výsledků do MAT souboru**

```
save('students.mat', 'T');
```

Proč: `MAT` je nativní binární formát MATLABu. Ukládá proměnné přímo tak, jak jsou v paměti (rychlé načítání, uchování typů).

9. **Načtení MAT souboru**

```
load('students.mat');
```

Proč: Proměnná `T` se načte zpět do workspace. Formát MAT je vhodný pro interní práci, méně vhodný pro sdílení mimo MATLAB.

10. **Uložení více proměnných do MAT souboru**

```
vek = T.Vek;
znamky = T.Znamka;
save('subset.mat','vek','znamky');
```

Proč: Do jednoho MAT souboru lze uložit více proměnných. Při načítání se vše vrátí zpět.

11. **Uložení tabulky do TXT (jednoduchý zápis)**

```
writetable(T, 'students.txt','Delimiter','\t');
```

Proč: Textový formát s tabulátorem (`\t`) se dá snadno otevřít v textovém editoru. Výhodné, pokud Excel není k dispozici.

12. **Zápis vybraných hodnot do TXT pomocí fprintf**

```
fid = fopen('names.txt','w');
for i=1:height(T)
    fprintf(fid,'%s má známku %d\n',T.Jmeno{i},T.Znamka(i));
end
fclose(fid);
```

Proč: Nízká úroveň zápisu – máme plnou kontrolu nad formátem řádků. Každý student se zapíše na jeden řádek.

13. **Čtení hodnot z TXT**

```
fid = fopen('names.txt','r');
C = textscan(fid,'%s %*s %*s %d');
fclose(fid);
```

Proč: `textscan` načte text podle formátu. Zde čteme jméno a známku (ostatní ignorujeme pomocí `%*s`).

14. **Přidání výsledků do Excelu na nový list**

```
writetable(T,'students.xlsx','Sheet','Data');
```

Proč: Umožňuje mít v Excelu více listů (např. data, výsledky).

15. **Export průměrné známky do TXT**

```
meanGrade = mean(T.Znamka);
fid = fopen('avg_grade.txt','w');
fprintf(fid,'Prumerna znamka: %.2f\n', meanGrade);
fclose(fid);
```

Proč: Ukládáme jednu statistiku do textového souboru, snadno čitelného i bez MATLABu.

16. **Načtení CSV s jiným oddělovačem (středník)**

```
opts = detectImportOptions('students.csv','Delimiter',';');
T_semicolon = readtable('students.csv',opts);
```

Proč: Některé CSV používají ; místo čárky. Pomocí `detectImportOptions` můžeme nastavit formát.

17. **Export filtrovaných dat do CSV**

```
T_good = T(T.Znamka<=2,:);
writetable(T_good,'good_students.csv');
```

Proč: Často chceme uložit jen část dat (např. studenti s dobrými známkami).

18. **Export do souboru s pevnou šířkou sloupců**

```
fid = fopen('fixedwidth.txt','w');
for i=1:height(T)
    fprintf(fid,'%-10s %3d %3d %3d %2d\n',T.Jmeno{i},T.Vek(i)
       ,T.Vyska(i),T.Vaha(i),T.Znamka(i));
end
fclose(fid);
```

Proč: Pevná šířka sloupců je starší formát, stále používaný v některých aplikacích. Zajišťuje zarovnání dat do sloupců.

19. **Načtení pevně formátovaného souboru**

```
fid = fopen('fixedwidth.txt','r');
C = textscan(fid,'%-10s %d %d %d %d');
fclose(fid);
```

Proč: Pomocí `textscan` se dá zpětně načíst i takto formátovaný text.

20. **Kombinace více formátů: načíst CSV, uložit MAT a TXT**

```
T = readtable('students.csv');
save('all_data.mat','T');
writetable(T,'all_data.txt','Delimiter','\t');
```

Proč: Praktická úloha – stejná data můžeme ukládat ve více formátech podle potřeby (MAT pro práci v MATLABu, TXT pro sdílení).

Kapitola 5

TEST po 1 měsíci

5.1 Test – Matice a vektory (1. týden)

Instrukce

Každá otázka má čtyři možnosti (A–D). Může být správně jedna nebo více možností. Za každou otázkou následuje správná odpověď a vysvětlení.

1. Jak vytvoříte vektor $v = [1, 2, 3, 4, 5]$?

```
A) v = [1 2 3 4 5];
B) v = 1:5;
C) v = linspace(1,5,5);
D) v = (1:5)';
```

   Správná odpověď: A, B, C. **Proč:** A přímo zapíše vektor, B použije operátor `:`, C využije `linspace`. Možnost D vytvoří sloupcový vektor, ne řádkový.

2. Jak získáte transponovaný vektor v^T z $v = [1, 2, 3]$?

```
A) v';
B) transpose(v);
C) v.';
D) trans(v);
```

   Správná odpověď: A, B, C. **Proč:** MATLAB zná `'` (konjugovaná transpozice), `.'` (čistá transpozice), `transpose`. Funkce `trans` neexistuje.

3. Jak vytvoříte jednotkovou matici I_3?

```
A) eye(3);
B) ones(3);
C) diag([1 1 1]);
D) speye(3);
```

   Správná odpověď: A, C, D. **Proč:** `eye(3)` i `speye(3)` tvoří jednotkovou matici. `diag([1 1 1])` vytvoří diagonální matici se stejnými prvky. `ones(3)` vytvoří matici samých jedniček.

4. Jak zjistíte počet řádků matice A?

```
A) size(A,1);
B) length(A);
C) height(A);
D) rows(A);
```

   Správná odpověď: A. **Proč:** `size(A,1)` vrací počet řádků. `length(A)` dává větší z rozměrů, `height` funguje jen na tabulky, `rows` není funkce.

5. Co provede příkaz `zeros(2,3)`?

```
A) Vytvoří vektor nul délky 2.
B) Vytvoří matici 2x3 nul.
C) Vytvoří matici 3x2 nul.
D) Chyba.
```

Správná odpověď: B. **Proč:** Funkce `zeros(m,n)` dává $m \times n$ nulovou matici.

6. Jak získáte poslední prvek vektoru v?

```
A) v(end);
B) v(length(v));
C) v(size(v,2));
D) v(last);
```

Správná odpověď: A, B, C. **Proč:** Index `end`, `length` i `size(v,2)` vrací poslední prvek. `last` není definováno.

7. Jak vytvořit matici $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$?

```
A) A = [1 2; 3 4];
B) A = [1,2;3,4];
C) A = [1 2 3 4];
D) A = [1:2;3:4];
```

Správná odpověď: A, B, D. **Proč:** Středník odděluje řádky. Varianta C vytvoří vektor.

8. Co provede příkaz `reshape(1:6,2,3)`?

```
A) Matice 2x3
B) Matice 3x2
C) Chyba
D) Sloupcový vektor délky 6
```

Správná odpověď: A. **Proč:** Funkce `reshape` plní matici po sloupcích, výsledek je 2×3.

9. Jak zjistit determinant matice A?

```
A) det(A);
B) determinant(A);
C) |A|;
D) prod(diag(A));
```

Správná odpověď: A. **Proč:** MATLAB zná pouze `det`. Ostatní možnosti neexistují nebo nejsou obecné.

10. Co vrátí `diag(1:3)`?

```
A) [1 2 3]
B) diag([1 2 3])
C) Matice s diagonálou 1,2,3
D) Chyba
```

Správná odpověď: C. **Proč:** Funkce `diag(v)` vrací diagonální matici z vektoru v.

11. Jak spočítat Frobeniovu normu $\|A\|_F$?

```
A) norm(A,'fro');
B) sqrt(sum(A.^2,'all'));
C) norm(A);
D) sum(abs(A),'all');
```

Správná odpověď: A, B, C. **Proč:** Ve výchozím nastavení `norm(A)` vrací Frobeniovu normu pro matice. Výpočet lze ověřit součtem čtverců všech prvků.

12. Jak ověříte, že matice A je symetrická?

```
A) issymmetric(A);
B) isequal(A,A');
C) issym(A);
D) A == A.';
```

Správná odpověď: A, B, D. **Proč:** Funkce `issymmetric` i `isequal` fungují. Operátor `==` vrátí logickou matici, kterou lze dále vyhodnotit.

13. Jak získáte stopu matice A (součet diagonály)?

```
A) trace(A);
B) sum(diag(A));
C) tr(A);
D) det(A);
```

Správná odpověď: A, B. **Proč:** MATLAB má funkci `trace`. Lze to spočítat i pomocí `diag` a `sum`. `tr` neexistuje.

14. Jak přidáte k matici A nový řádek $[7, 8, 9]$?

```
A) A = [A;7 8 9];
B) A(end+1,:) = [7 8 9];
C) append(A,[7 8 9]);
D) A = vertcat(A,[7 8 9]);
```

Správná odpověď: A, B, D. **Proč:** Středník nebo `vertcat` spojí po řádcích. Indexování `end+1` funguje pro rozšíření.

15. Jak vytvoříte vektor rovnoměrně rozmístěných 11 bodů v intervalu $[0, 10]$?

```
A) linspace(0,10,11);
B) 0:1:10;
C) 0:10;
D) range(0,10,11);
```

Správná odpověď: A, B, C. **Proč:** `linspace(0,10,11)` i zápisy s dvojtečkou vytvoří požadovaný vektor. `range` v MATLABu neexistuje.

5.2 Test – Funkce a skripty (2. týden)

Instrukce

Každá otázka má čtyři možnosti (A–D). Může být správně jedna nebo více možností. Za každou otázkou následuje správná odpověď a vysvětlení.

1. Jaký je rozdíl mezi skriptem a funkcí v MATLABu?

```
A) Skript má svůj vlastní workspace.
B) Funkce má svůj vlastní workspace.
C) Skript nemá vstupní/výstupní argumenty.
D) Funkce může mít vstupní i výstupní argumenty.
```

   Správná odpověď: B, C, D. **Proč:** Skripty sdílí workspace s uživatelem a neberou vstupy ani výstupy. Funkce mají oddělený workspace a mohou mít argumenty.

2. Jak správně definovat funkci v souboru `myfun.m`?

```
A) function y = myfun(x)
B) function = myfun(x) y
C) def myfun(x): return y
D) function myfun(x)
```

   Správná odpověď: A. **Proč:** Funkce musí začínat klíčovým slovem `function`. Pythonová syntaxe (C) není MATLAB. D je neúplné.

3. Jak zavoláte funkci uloženou v `myfun.m` s jedním vstupem?

```
A) y = myfun(x);
B) myfun(x);
C) y = myfun x;
D) call myfun(x);
```

   Správná odpověď: A, B. **Proč:** Funkci lze zavolat s nebo bez přiřazení výsledku. Ostatní nejsou platná MATLAB volání.

4. Co provede příkaz `nargin` uvnitř funkce?

```
A) Vrátí počet argumentů funkce při definici.
B) Vrátí počet argumentů, se kterými byla funkce volána.
C) Vrátí počet výstupů funkce.
D) Vrátí počet lokálních proměnných.
```

   Správná odpověď: B. **Proč:** `nargin` dává skutečný počet vstupů při volání, ne při definici.

5. Co provede příkaz `varargout`?

```
A) Umožní funkci přijmout proměnný počet vstupů.
B) Umožní funkci vrátit proměnný počet výstupů.
C) Vytvoří globální proměnnou.
D) Je totéž co varargin.
```

Správná odpověď: B. **Proč:** `varargout` slouží pro libovolný počet výstupů. `varargin` je na vstupy.

6. Kde musí být definována funkce uvnitř skriptu, aby fungovala v MATLABu (od R2016b)?

```
A) Na začátku skriptu před kódem.
B) Uprostřed skriptu.
C) Na konci skriptu, po hlavním kódu.
D) Nemůže být ve skriptu.
```

Správná odpověď: C. **Proč:** Funkce mohou být definovány až na konci skriptu, nikoli uprostřed.

7. Jak vyhlásíte proměnnou jako globální?

```
A) global x
B) Global x
C) x = global;
D) global(x)
```

Správná odpověď: A. **Proč:** Používá se klíčové slovo `global` následované názvem proměnné.

8. Co provede příkaz `clear` ve skriptu?

```
A) Smaže všechny proměnné z workspace.
B) Smaže proměnné jen z funkce.
C) Zavře všechny grafy.
D) Resetuje MATLAB.
```

Správná odpověď: A. **Proč:** `clear` odstraňuje proměnné z workspace. Grafy zůstávají, MATLAB se nerestartuje.

9. Jakým příkazem uložíte proměnné do souboru MAT?

```
A) save('data.mat')
B) store('data.mat')
C) save data.mat
D) writemat('data.mat')
```

Správná odpověď: A, C. **Proč:** Funkce `save` umí oba zápisy. Ostatní nejsou MATLAB funkce.

10. Co provede příkaz `exist('myfun','file')`?

```
A) Ověří, zda existuje soubor myfun.m.
B) Vrátí true, pokud existuje proměnná myfun.
C) Vrátí číslo reprezentující typ objektu.
D) Zavolá funkci myfun.
```

Správná odpověď: A, C. **Proč:** `exist` vrací číslo podle typu (soubor, proměnná, vestavěná funkce). Nevolá funkci.

11. Jak spustíte skript `myscript.m`?

```
A) run myscript
B) run('myscript')
C) myscript
D) execute myscript
```

Správná odpověď: A, B, C. **Proč:** Skript lze spustit příkazem `run` nebo prostým napsáním jména. `execute` není MATLAB příkaz.

12. Jak zjistíte cestu k souboru `myscript.m`?

```
A) which myscript
B) path myscript
C) locate('myscript')
D) which('myscript')
```

Správná odpověď: A, D. **Proč:** Příkaz `which` ukazuje, odkud MATLAB soubor načte. `path` vypíše jen cestu k adresářům.

13. Jak zjistíte počet výstupů, s nimiž byla funkce volána?

```
A) nargout
B) nargin
C) outputs
D) numout
```

Správná odpověď: A. **Proč:** `nargout` uvnitř funkce vrací počet očekávaných výstupů. `nargin` je pro vstupy.

14. Co provede příkaz `return` ve funkci?

```
A) Ukončí funkci okamžitě.
B) Vrátí hodnotu bez ohledu na výstupní proměnné.
C) Restartuje funkci od začátku.
D) Ukončí MATLAB.
```

Správná odpověď: A. **Proč:** `return` okamžitě ukončí běh funkce, ale nevynutí návrat hodnot.

15. Jak vypsat nápovědu k funkci `plot`?

```
A) help plot
B) doc plot
C) ?plot
D) info plot
```

Správná odpověď: A, B. **Proč:** `help` vypíše text v příkazovém okně, `doc` otevře dokumentaci v prohlížeči. `?` není MATLAB syntaxe, `info` není platné.

5.3 Test – Grafy a vizualizace (3. týden)

Instrukce

Každá otázka má čtyři možnosti (A–D). Může být správně jedna nebo více možností. Za každou otázkou následuje správná odpověď a vysvětlení.

1. Jak vytvořit základní 2D graf funkce $y = \sin(x)$?

```
A) x = 0:0.1:10; plot(x, sin(x))
B) x = 0:0.1:10; plot(sin(x))
C) plot(sin(0:0.1:10))
D) fplot(@sin, [0, 10])
```

   Správná odpověď: A, D. **Proč:** A ručně vytvoří osu x a vykreslí $\sin(x)$. D použije automatické vzorkování. B a C chybí osa x.

2. Jak přidat popisky os?

```
A) xlabel('čas [s]'); ylabel('amplituda')
B) labelx('čas [s]'); labely('amplituda')
C) set(gca, 'xlabel', 'čas [s]')
D) xlabel čas [s]; ylabel amplituda
```

   Správná odpověď: A. **Proč:** Funkce `xlabel`, `ylabel` přidají popisky. Ostatní nejsou správná MATLAB syntaxe.

3. Jak zobrazit více křivek v jednom grafu?

```
A) plot(x, sin(x), x, cos(x))
B) plot(x, sin(x)); plot(x, cos(x))
C) hold on; plot(x, sin(x)); plot(x, cos(x)); hold off
D) plot([sin(x), cos(x)])
```

   Správná odpověď: A, C. **Proč:** A umožní více datových sad v jednom volání. C využívá `hold on`. B přepíše první graf. D je špatná syntaxe.

4. Jak vytvořit rozptylový graf (scatter plot)?

```
A) scatter(x, y)
B) plot(x, y, 'o')
C) bar(x, y)
D) scatterplot(x, y)
```

   Správná odpověď: A, B. **Proč:** Funkce `scatter` i `plot(...,'o')` vytvoří body. `bar` dělá sloupcový graf. `scatterplot` je pro komunikace toolbox.

5. Jak nastavit rozsah osy x?

```
A) xlim([0 5])
B) axis([0 5 -1 1])
C) set(gca,'XLim',[0 5])
D) set(gcf,'XLim',[0 5])
```

Správná odpověď: A, B, C. **Proč:** `xlim`, `axis`, nebo vlastnost osy. D je špatně – `gcf` je figura, ne osa.

6. Jak přidat legendu?

```
A) legend('sin','cos')
B) legend(['sin','cos'])
C) addlegend('sin','cos')
D) legend sin cos
```

Správná odpověď: A. **Proč:** Funkce `legend` s řetězci vytvoří popisky. Ostatní nejsou platná syntaxe.

7. Jak vytvořit 3D graf povrchu $z = \sin(x)\cos(y)$?

```
A) [X,Y] = meshgrid(-2:0.1:2); Z = sin(X).*cos(Y); surf(X,Y,Z
   )
B) [X,Y] = meshgrid(-2:0.1:2); Z = sin(X).*cos(Y); mesh(X,Y,Z
   )
C) [X,Y] = ndgrid(-2:0.1:2); Z = sin(X).*cos(Y); contour(X,Y,
   Z)
D) surf(sin(x).*cos(y))
```

Správná odpověď: A, B. **Proč:** `surf`, `mesh` vykreslí 3D plochu. Contour je 2D řez. D nefunguje, protože není definovaná mřížka.

8. Jak přidat název grafu?

```
A) title('Moje data')
B) plot.title('Moje data')
C) set(gca,'title','Moje data')
D) addtitle('Moje data')
```

Správná odpověď: A. **Proč:** Používá se funkce `title`. Ostatní jsou špatně.

9. Jak zobrazit mřížku v grafu?

```
A) grid on
B) grid off
C) set(gca,'XGrid','on','YGrid','on')
D) showgrid()
```

Správná odpověď: A, B, C. **Proč:** Grid lze zapnout/vypnout nebo nastavit přímo na osách. `showgrid` není MATLAB.

10. Jak vytvořit subplot 2×2 a aktivovat druhý panel?

```
A) subplot(2,2,2)
B) tiledlayout(2,2); nexttile(2)
C) subplot(2,2,1); hold on
D) plot(2,2,2)
```

Správná odpověď: A, B. **Proč:** `subplot` i `tiledlayout+nexttile` slouží k vícenásobným grafům. C jen drží graf v prvním subplotu. D je nesmysl.

11. Jak zobrazit histogram dat?

```
A) histogram(data)
B) hist(data)
C) bar(data)
D) plot(histcounts(data))
```

Správná odpověď: A, B. **Proč:** `histogram` je moderní, `hist` starší. `bar` by bral páry (x,y). `histcounts` vrací jen čísla.

12. Jak uložit graf do PNG?

```
A) saveas(gcf,'graf.png')
B) print('graf','-dpng')
C) exportgraphics(gca,'graf.png')
D) save('graf.png')
```

Správná odpověď: A, B, C. **Proč:** Tyto funkce exportují obrázky. `save` ukládá proměnné do MAT souboru.

13. Jak vytvořit animaci bodu pohybujícího se po kružnici?

```
A) for t=0:0.1:2*pi, plot(cos(t),sin(t),'o'); pause(0.1); end
B) for t=0:0.1:2*pi, scatter(cos(t),sin(t)); end
C) for t=0:0.1:2*pi, plot3(cos(t),sin(t),t); pause(0.1); end
D) comet(cos(0:0.1:2*pi),sin(0:0.1:2*pi))
```

Správná odpověď: A, D. **Proč:** A postupně kreslí body s pauzou. D funkce `comet` animuje trajektorii. B překresluje bez pauzy, C je 3D spirála.

14. Jak přidat text do grafu na souřadnice $(1, 0.5)$?

```
A) text(1,0.5,'Bod A')
B) annotation('textbox',[1,0.5,0.1,0.1],'String','Bod A')
C) addtext(1,0.5,'Bod A')
D) gtext('Bod A')
```

Správná odpověď: A, B, D. **Proč:** `text` umístí do datových souřadnic, `annotation` do relativních, `gtext` interaktivně kliknutím. `addtext` není MATLAB.

15. Jak vytvořit 3D konturový graf?

```
A) contour3(X,Y,Z)
B) contour(X,Y,Z, 'ShowText','on')
C) surf(X,Y,Z)
D) contourf(X,Y,Z)
```

Správná odpověď: A. **Proč:** `contour3` vykreslí kontury ve 3D prostoru. `contour` a `contourf` jsou 2D. `surf` vykreslí plochu, ne kontury.

5.4 Test – Práce s daty (4. týden)

Instrukce

Každá otázka má čtyři možnosti (A–D). Může být správně jedna nebo více možností. Za každou otázkou následuje správná odpověď a vysvětlení.

1. Jak načíst data z CSV souboru `students.csv`?

```
A) data = readmatrix('students.csv')
B) data = readtable('students.csv')
C) load('students.csv')
D) data = csvread('students.csv')
```

Správná odpověď: A, B, D. **Proč:** `readmatrix` a `readtable` jsou moderní a vhodné. `csvread` funguje, ale je zastaralé. `load` CSV neumí.

2. Jak vybrat studenty s věkem větším než 20 let ze tabulky?

```
A) T(T.Vek > 20, :)
B) T(T.Age > 20)
C) find(T.Vek > 20)
D) T(T.Vek > 20)
```

Správná odpověď: A, D. **Proč:** Řezání tabulky pomocí logického indexu. `find` vrátí indexy, nikoliv řádky. B nefunguje, pokud se sloupec jmenuje jinak.

3. Jak uložit tabulku `T` do Excelu?

```
A) writetable(T, 'soubor.xlsx')
B) save('soubor.xlsx','T')
C) xlswrite('soubor.xlsx', T)
D) writematrix(T, 'soubor.xlsx')
```

Správná odpověď: A, C. **Proč:** `writetable` je moderní, `xlswrite` starší. `save` ukládá jen do `.mat`, `writematrix` ztrácí názvy proměnných.

4. Jak spočítat průměr známek ve sloupci `Znamka`?

```
A) mean(T.Znamka)
B) average(T.Znamka)
C) mean(T{:,'Znamka'})
D) sum(T.Znamka)/length(T.Znamka)
```

Správná odpověď: A, C, D. **Proč:** `mean` funguje přímo na tabulku. D je ruční výpočet. `average` není MATLAB funkce.

5. Jak zjistit maximální hodnotu věku?

```
A) max(T.Vek)
B) maximum(T.Vek)
C) sort(T.Vek,'descend'); T.Vek(1)
D) [M,I] = max(T.Vek)
```

Správná odpověď: A, C, D. **Proč:** `max` je přímo, C ručně přes třídění, D vrátí i index. `maximum` neexistuje.

6. Jak vypsat počet řádků tabulky?

```
A) height(T)
B) size(T,1)
C) length(T)
D) numel(T)
```

Správná odpověď: A, B. **Proč:** `height` dává počet řádků. `size(T,1)` totéž. `length` je nespolehlivé. `numel` dává celkový počet prvků.

7. Jak přidat nový sloupec do tabulky?

```
A) T.Prijmeni = {'Novak','Svoboda'}
B) T(:, 'Prijmeni') = {'Novak','Svoboda'}
C) T = addvars(T, {'Novak','Svoboda'}, 'NewVariableNames','
   Prijmeni')
D) T = [T 'Novak']
```

Správná odpověď: A, C. **Proč:** A přiřadí přímo, C používá `addvars`. B není validní syntaxe. D je nesprávná operace.

8. Jak vytvořit histogram hodnot známek?

```
A) histogram(T.Znamka)
B) hist(T.Znamka)
C) bar(T.Znamka)
D) plot(histcounts(T.Znamka))
```

Správná odpověď: A, B. **Proč:** `histogram` je moderní, `hist` funguje. `bar` by zobrazil jednotlivé hodnoty, ne rozdělení.

9. Jak provést lineární regresi mezi věkem a známkou?

```
A) polyfit(T.Vek, T.Znamka, 1)
B) regress(T.Znamka, [ones(size(T.Vek)) T.Vek])
C) fitlm(T,'Znamka ~ Vek')
D) lsqcurvefit(@(b,x) b(1)*x+b(2), [1 0], T.Vek, T.Znamka)
```

Správná odpověď: A, B, C, D. **Proč:** Všechny uvedené přístupy jsou platné – liší se metodou a výstupem.

10. Jak najít chybějící hodnoty v tabulce?

```
A) ismissing(T)
B) T(ismissing(T), :)
C) find(isnan(T))
D) T(isnan(T.Znamka), :)
```

Správná odpověď: A, B, D. **Proč:** `ismissing` detekuje NaN/NaT/prázdné. `isnan` funguje jen na číselné sloupce.

11. Jak uložit data do formátu MAT?

```
A) save('data.mat','T')
B) writetable(T,'data.mat')
C) save T data.mat
D) save('data','T')
```

Správná odpověď: A. **Proč:** `save` do MAT ukládá proměnné. Ostatní nejsou validní.

12. Jak pracovat s datem a časem v MATLABu?

```
A) datetime('now')
B) now()
C) clock()
D) date()
```

Správná odpověď: A, B, C, D. **Proč:** Všechny možnosti jsou platné: `datetime` je objekt, `now` dává číslo, `clock` vektor, `date` řetězec.

13. Jak resamplovat časová data na denní průměr?

```
A) retime(T,'daily','mean')
B) groupsummary(T,'Time','day','mean')
C) resample(T,'daily')
D) mean(T, 'daily')
```

Správná odpověď: A, B. **Proč:** `retime` a `groupsummary` fungují s časem. C a D nejsou správná syntaxe.

14. Jak uložit výsledky výpočtů do CSV?

```
A) writetable(T,'vysledky.csv')
B) save('vysledky.csv','T')
C) writematrix(T.Znamka,'znamky.csv')
D) exportgraphics(T,'vysledky.csv')
```

Správná odpověď: A, C. **Proč:** `writetable` zachová názvy, `writematrix` uloží jen čísla. `save` a `exportgraphics` nefungují pro CSV.

15. Jak zobrazit korelaci mezi dvěma sloupci?

```
A) corrcoef(T.Vek, T.Znamka)
B) correlation(T.Vek, T.Znamka)
C) corr(T{:,'Vek'}, T{:,'Znamka'})
D) corrplot(T)
```

Správná odpověď: A, C, D. **Proč:** `corrcoef`, `corr`, i `corrplot` fungují. `correlation` není MATLAB funkce.

Část II

2. Měsíc - Efektivní programování Cíl : psát rychlý, čitelný a znovupoužitelný kód, umět profilovat výkon a optimalizovat.

Kapitola 6

5. Týden Funkce na profi úrovni

6.1 Pondělí – Funkce s `varargin`, `varargout`

Cíl dne: Naučit se psát funkce, které přijímají libovolný počet vstupů a mohou vracet více výstupů. To umožňuje větší flexibilitu, znovupoužitelnost a čitelnost kódu.

Úkoly a řešení

1. Definujte jednoduchou funkci s `varargin`, která jen vypíše počet vstupů.

```
function countInputs(varargin)
    fprintf('Pocet vstupu: %d\n', nargin);
end
```

   Proč: `nargin` vrací počet vstupů. `varargin` se chová jako cell array.

2. Funkce, která vypíše všechny vstupy.

```
function showInputs(varargin)
    for k = 1:nargin
        disp(varargin{k});
    end
end
```

   Proč: Přístup `varargin{k}` vybírá jednotlivé vstupy z cell array.

3. Funkce, která sečte libovolný počet čísel.

```
function s = sumAll(varargin)
    s = 0;
    for k = 1:nargin
        s = s + varargin{k};
    end
end
```

   Proč: Obecná alternativa `sum`, když nevíme předem počet vstupů.

4. Funkce, která vrací více výstupů pomocí `varargout`.

```
function varargout = basicOps(a,b)
    varargout{1} = a+b;
    varargout{2} = a-b;
    varargout{3} = a*b;
end
```

   Proč: Uživatel si může vybrat, kolik výstupů využije.

5. Zavolejte předchozí funkci s různým počtem výstupů.

```
x = 10; y = 3;
soucet = basicOps(x,y);
[soucet,rozdil] = basicOps(x,y);
[soucet,rozdil, soucin] = basicOps(x,y);
```

   Proč: MATLAB ignoruje nevyžádané výstupy.

6. Funkce, která vypočítá různé statistiky dle volby.

```
function varargout = stats(data, varargin)
    for k = 1:length(varargin)
        switch varargin{k}
            case 'mean'
                varargout{k} = mean(data);
            case 'std'
                varargout{k} = std(data);
            case 'median'
                varargout{k} = median(data);
            case 'max'
                varargout{k} = max(data);
            otherwise
                error('Neznamy prikaz');
        end
    end
end
```

Proč: Flexibilní funkce – uživatel si vybere, které statistiky potřebuje.

7. Ukázka použití funkce `stats`.

```
data = [1,2,3,4,5,10];
[m,s] = stats(data, 'mean', 'std');
```

Proč: Vrátí jen to, co chceme – tady průměr a směrodatnou odchylku.

8. Přidání volitelného parametru pro robustní průměr.

```
[m,med] = stats(data,'mean','median');
```

Proč: Medián je odolnější proti odlehlým hodnotám.

9. Funkce, která přijme libovolný počet vektorů a vrátí jejich délky.

```
function varargout = vecLengths(varargin)
    for k = 1:nargin
        varargout{k} = length(varargin{k});
    end
end
```

Proč: Dynamické přizpůsobení počtu výstupů počtu vstupů.

10. Funkce, která pro každý vstupní vektor spočítá průměr.

```
function varargout = means(varargin)
    for k = 1:nargin
        varargout{k} = mean(varargin{k});
    end
end
```

Proč: Vhodné pro srovnání více datasetů najednou.

11. Funkce, která kombinuje volitelné parametry.

```
function r = polyEval(x, varargin)
    r = 0;
    for k = 1:length(varargin)
        r = r + varargin{k}*x^(k-1);
    end
end
```

Proč: Hodnotí polynom zadaný koeficienty – libovolná délka.

12. Funkce, která přijme čísla a vrátí minimum a maximum.

```
function varargout = minmax(varargin)
    vals = cell2mat(varargin);
    varargout{1} = min(vals);
    varargout{2} = max(vals);
end
```

Proč: Ukázka převodu cell → matice pomocí `cell2mat`.

13. Funkce, která otestuje, zda jsou všechny vstupy skaláry.

```
function tf = allScalars(varargin)
    tf = all(cellfun(@isscalar, varargin));
end
```

Proč: `cellfun` elegantně aplikuje funkci na každý prvek.

14. Funkce, která přijme libovolný počet matic a spočítá jejich součet.

```
function S = sumMatrices(varargin)
    S = 0;
    for k = 1:nargin
        S = S + varargin{k};
    end
end
```

Proč: Snadné rozšíření pro více operandů.

15. Funkce, která kontroluje vstupy a vyhodí chybu, pokud nejsou čísla.

```
function s = safeSum(varargin)
    s = 0;
    for k = 1:nargin
        if ~isnumeric(varargin{k})
            error('Vstup musi byt cislo');
        end
        s = s + varargin{k};
    end
end
```

Proč: Obrana proti chybám uživatele.

16. Funkce, která se chová jinak podle počtu výstupů.

```
function varargout = quadratic(a,b,c)
    D = b^2-4*a*c;
    r1 = (-b+sqrt(D))/(2*a);
    r2 = (-b-sqrt(D))/(2*a);
    varargout{1} = r1;
    if nargout > 1
        varargout{2} = r2;
    end
end
```

Proč: Pokud chceme jen jeden kořen, není třeba počítat druhý.

17. Funkce, která loguje své vstupy a výstupy.

```
function varargout = logFunction(varargin)
    disp('Vstupy:'); disp(varargin);
    varargout = varargin; % vratit stejne
    disp('Vystupy:'); disp(varargout);
end
```

Proč: Užitečné při ladění kódu.

18. Funkce, která pro každý vstupní vektor spočítá součet i průměr.

```
function varargout = sumAndMean(varargin)
    for k = 1:nargin
        varargout{k} = [sum(varargin{k}), mean(varargin{k})];
    end
end
```

Proč: Vrací dvojici hodnot – ukázka flexibility `varargout`.

19. Funkce, která využívá defaultní hodnotu, pokud vstup není zadaný.

```
function y = squareOrDefault(varargin)
    if nargin < 1
        x = 2;
    else
        x = varargin{1};
    end
    y = x^2;
end
```

Proč: Uživatel nemusí vždy zadat vstup.

20. Funkce, která kombinuje `varargin` a `varargout` do univerzální kalkulačky.

```
function varargout = calc(op, varargin)
    switch op
        case 'add'
            varargout{1} = sum(cell2mat(varargin));
        case 'multiply'
            varargout{1} = prod(cell2mat(varargin));
```

```
7           case 'stats'
8               data = cell2mat(varargin);
9               varargout{1} = mean(data);
10              if nargout>1, varargout{2}=std(data); end
11      end
12  end
```

Proč: Ukázka univerzálního designu funkce – podle volby `op` provádí různé operace.

6.2 Úterý – Validace vstupů a profilování výkonu

Cíl dne: Naučit se, jak psát bezpečnější a rychlejší kód – tedy jak ošetřit vstupy funkcí a jak měřit jejich výkonnost.

6.2.1 Úkoly a řešení

1. Použití `narginchk` pro omezení počtu vstupů.

```
function y = addTwo(a,b)
    narginchk(2,2); % presne dva vstupy
    y = a+b;
end
```

Proč: `narginchk(min,max)` zaručí, že funkce byla volána se správným počtem vstupů.

2. Povolení volitelného parametru.

```
function y = addDefault(a,b)
    narginchk(1,2);
    if nargin < 2, b=0; end
    y = a+b;
end
```

Proč: Pokud není zadán druhý vstup, nastavíme výchozí hodnotu.

3. Validace, že vstup je číslo.

```
function y = squareNumber(x)
    validateattributes(x, {'numeric'}, {'scalar'});
    y = x^2;
end
```

Proč: `validateattributes` kontroluje datový typ i vlastnosti.

4. Validace, že matice obsahuje jen nenulové prvky.

```
function y = safeInverse(A)
    validateattributes(A, {'numeric'}, {'nonzero'});
    y = inv(A);
end
```

Proč: Vlastnost `'nonzero'` vynutí, že žádný prvek není 0 (užitečné např. pro diagonální matice).

5. Validace rozměrů vstupů.

```
function y = addVectors(a,b)
    validateattributes(a, {'numeric'}, {'vector'});
    validateattributes(b, {'numeric'}, {'vector'});
    assert(length(a)==length(b),'Rozdilne delky');
    y = a+b;
end
```

Proč: `assert` doplňuje validaci, když chceme vlastní podmínku.

6. Validace kladných čísel.

```
function r = circleArea(r)
    validateattributes(r, {'numeric'}, {'scalar','positive'})
        ;
    r = pi*r^2;
end
```

Proč: Ošetříme, aby poloměr nebyl záporný.

7. Kombinace více validací.

```
function m = safeMean(x)
    validateattributes(x, {'numeric'}, {'vector','nonempty'})
        ;
    m = mean(x);
end
```

Proč: Vstup musí být neprázdný vektor.

8. Validace matic s celými čísly.

```
function s = sumIntegers(A)
    validateattributes(A, {'numeric'}, {'integer'});
    s = sum(A,'all');
end
```

Proč: Hodí se např. pro práci s indexy.

9. Validace velikosti.

```
function d = determinant3x3(A)
    validateattributes(A, {'numeric'}, {'size',[3,3]});
    d = det(A);
end
```

Proč: Umožňuje explicitně vyžadovat rozměr.

10. Validace logických vstupů.

```
function r = toggleSwitch(x)
    validateattributes(x, {'logical'}, {'scalar'});
    r = ~x;
end
```

Proč: Funkce smí přijmout jen logickou hodnotu.

11. Měření času výpočtu.

```
tic
pause(0.5)
elapsed = toc
```

Proč: `tic/toc` měří čas mezi zavoláním.

12. Porovnání rychlosti cyklu a vektorizace.

```
n=1e5;
tic; for k=1:n, a(k)=k^2; end; t1=toc;
tic; b=(1:n).^2; t2=toc;
```

Proč: Vektorizace bývá mnohem rychlejší než for-cyklus.

13. Profilování funkce.

```
profile on
magic(1000);
profile viewer
```

Proč: `profile` měří časovou náročnost po jednotlivých funkcích.

14. Vyhodnocení funkce pomocí profilu.

```
profile on
for k=1:1000, sin(k); end
profile off
stats = profile('info')
```

Proč: Získáme strukturu s podrobnými informacemi o výkonu.

15. Porovnání algoritmů – hledání maxima.

```
x = rand(1e6,1);
tic; m1=max(x); t1=toc;
tic; m2=x(1); for k=2:length(x), if x(k)>m2, m2=x(k); end,
   end; t2=toc;
```

Proč: Ukazuje rozdíl mezi vestavěnou funkcí a implementací smyčkou.

16. Validace typu string.

```
function greet(name)
    validateattributes(name, {'char','string'}, {'nonempty'})
       ;
    disp(['Ahoj, ', char(name)]);
end
```

Proč: Funkce akceptuje řetězec v obou formátech.

17. Validace čtvercové matice s kladnými prvky.

```
function y = sumPositive(A)
    validateattributes(A, {'numeric'}, {'square','positive'})
       ;
    y = sum(A,'all');
end
```

Proč: Zajistí, že všechny prvky > 0 a matice je čtvercová.

18. Porovnání dvou variant výpočtu faktoriálu.

```
n=500;
tic; f1=prod(1:n); t1=toc;
tic; f2=1; for k=1:n, f2=f2*k; end; t2=toc;
```

Proč: Ukazuje, že vektorové operace bývají rychlejší.

19. Profilování uživatelské funkce.

```
function y = slowFunc(n)
    y=0;
    for k=1:n
        y=y+sqrt(k);
    end
end

profile on
slowFunc(1e5);
profile viewer
```

Proč: Identifikujeme části kódu, které nejvíce zatěžují CPU.

20. Kombinace validace a měření výkonu.

```
function m = meanSafe(x)
    validateattributes(x, {'numeric'}, {'vector','nonempty'})
       ;
    tic
    m = mean(x);
    fprintf('Cas vypoctu: %.6f s\n', toc);
end
```

Proč: Ukazuje, jak spojit ošetření vstupů i měření efektivity.

6.3 Středa - Defaultní argumenty a prealokace paměti

Cíl dne: Naučit se, jak psát flexibilní funkce s defaultními hodnotami pomocí `inputParser` a jak efektivně pracovat s velkými daty, aby kód běžel rychleji (prealokace, vektorizace).

6.3.1 Úkoly a řešení

1. Vytvoření funkce s defaultním argumentem pomocí inputParser.

```
function y = multiply(a,b)
    p = inputParser;
    addRequired(p,'a',@isnumeric)
    addOptional(p,'b',2,@isnumeric) % default 2
    parse(p,a,varargin{:});
    y = p.Results.a * p.Results.b;
end
```

Proč: `inputParser` umožňuje definovat povinné a volitelné argumenty, kontrolu typů a defaultní hodnoty.

2. Volání funkce s a bez volitelného parametru.

```
multiply(5)     % použije b=2
multiply(5,3)   % použije b=3
```

Proč: Funkce se chová flexibilně podle vstupu, výchozí hodnoty zajišťují bezpečné volání.

3. Prealokace vektoru před cyklem.

```
n=1e6;
x=zeros(1,n); % prealokace
for k=1:n
    x(k)=k^2;
end
```

Proč: Prealokace paměti výrazně zrychluje smyčky – zabraňuje opakovanému realokování.

4. Porovnání s dynamickým růstem.

```
y=[];
tic
for k=1:n, y(k)=k^2; end
toc
```

Proč: Dynamické přidávání prvků je výrazně pomalejší – Matlab musí každou iteraci přidělovat novou paměť.

5. Vektorizace operací místo cyklu.

```
x = (1:n).^2;
```

Proč: Vektorové operace jsou v MATLABu optimalizované, běží mnohem rychleji než for-cyklus.

6. Prealokace matice.

```
A = zeros(1000,1000);
for i=1:1000
    for j=1:1000
        A(i,j) = i+j;
    end
end
```

Proč: Opět zamezuje opakovanému rozšiřování matice během smyčky.

7. Použití prealokace pro dynamický výpočet.

```
n=1000;
x=linspace(0,2*pi,n);
y=zeros(1,n);
for k=1:n
    y(k)=sin(x(k));
end
```

Proč: Vektor x je již definován, y je prealokováno – efektivní výpočet sinusových hodnot.

8. Funkce s více volitelnými parametry a inputParser.

```
function res = compute(a,varargin)
    p = inputParser;
    addRequired(p,'a',@isnumeric)
    addParameter(p,'scale',1,@isnumeric)
    addParameter(p,'shift',0,@isnumeric)
    parse(p,a,varargin{:})
    res = a*p.Results.scale + p.Results.shift;
end
```

Proč: `addParameter` definuje volitelné pojmenované argumenty s defaultní hodnotou a kontrolou typu.

9. Prealokace buněk (cell arrays).

```
C = cell(1,100);
for k=1:100
    C{k} = rand(3);
end
```

Proč: Předem vytvořená cell array zrychluje cyklus a zabraňuje opakované alokaci.

10. Dynamické pole vs. prealokace

```
D={};
tic
for k=1:100, D{k}=rand(3); end
toc
```

Proč: Ukazuje, že i u cell arrays je prealokace efektivnější.

11. Funkce s volitelným parametrem a kontrolou typu string.

```
function greet(name,varargin)
    p = inputParser;
    addRequired(p,'name',@(x) ischar(x) || isstring(x))
    addParameter(p,'punct','!',@ischar)
    parse(p,name,varargin{:})
    disp([char(p.Results.name) p.Results.punct])
end
```

Proč: Flexibilní funkce s defaultním parametrem a kontrolou vstupu.

12. Prealokace pro velký datový soubor.

```
n=1e6;
data=zeros(n,3); % 3 sloupce
for k=1:n
    data(k,:)=[k,k^2,sqrt(k)];
end
```

Proč: Matice je velká, prealokace zabraňuje opakovanému realokování.

13. Použití vektorizace pro stejný výsledek.

```
k=(1:n)';
data=[k,k.^2,sqrt(k)];
```

Proč: Vektorový zápis je rychlejší a elegantnější.

14. Prealokace při ukládání výsledků do struktury.

```
S(n).val = [];
for k=1:n
    S(k).val = k^2;
end
```

Proč: Předem vytvořená struktura zrychluje zápis do polí.

15. Využití inputParser s více volitelnými argumenty.

```
function res = scaleShift(a,varargin)
    p = inputParser;
    addRequired(p,'a',@isnumeric)
    addParameter(p,'scale',1,@isnumeric)
    addParameter(p,'shift',0,@isnumeric)
    parse(p,a,varargin{:})
    res = (a + p.Results.shift)*p.Results.scale;
end
```

Proč: Kombinuje posun a škálování s defaultními hodnotami.

16. Prealokace pro kumulativní součet.

```
n=1e6;
cumsum_vec=zeros(1,n);
for k=1:n
    cumsum_vec(k)=sum(1:k);
end
```

Proč: Prealokace zrychluje smyčku.

17. Porovnání rychlosti funkce s for-cyklem.

```
tic; res=cumsum(1:n); toc
tic; res2=zeros(1,n); for k=1:n, res2(k)=sum(1:k); end; toc
```

Proč: Vestavěná funkce je výrazně rychlejší než vlastní cyklus.

18. Prealokace matice pro animaci.

```
n=500;
X=zeros(1,n);
for k=1:n
    X(k)=sin(2*pi*k/n);
end
```

Proč: U animací se doporučuje prealokovat všechny datové struktury.

19. Využití inputParser pro logické volby.

```
function plotData(x,varargin)
    p = inputParser;
    addRequired(p,'x',@isnumeric)
    addParameter(p,'logPlot',false,@islogical)
    parse(p,x,varargin{:})
    if p.Results.logPlot
        semilogy(x)
    else
        plot(x)
    end
end
```

Proč: Funkce je flexibilní, umožňuje volbu typu grafu.

20. Prealokace pro práci s více datovými soubory.

```
nFiles=100;
dataCell=cell(1,nFiles); % prealokace
for k=1:nFiles
    dataCell{k}=rand(100,3);
end
```

Proč: Velké množství dat je bezpečně a rychle uloženo do prealokovaných buněk.

6.4 Čtvrtek - Funkce vracející struct a cell array, vizualizace statistik, export dat, korelace a regrese

Cíl dne: Naučit se psát funkce, které vracejí různé formáty výstupu (struct, cell array), dále procvičit vizualizaci statistik, export dat a provést základní analýzu vztahů mezi veličinami (pomocí korelace a lineární regrese).

Úkoly a řešení

1. Funkce vracející strukturu se souhrnnými statistikami.

```
function S = statsStruct(x)
    S.mean = mean(x);
    S.std = std(x);
    S.min = min(x);
    S.max = max(x);
end
```

   Proč: Struktura je vhodná pro pojmenované ukládání výsledků, což zvyšuje čitelnost.

2. Použití funkce na vektor dat.

```
data = randn(1,100);
S = statsStruct(data);
```

   Proč: Výsledky jsou uloženy v přehledném formátu `S.mean`, `S.std` atd.

3. Funkce vracející cell array se statistikami.

```
function C = statsCell(x)
    C = {mean(x), std(x), min(x), max(x)};
end
```

   Proč: Cell array je vhodná, když pořadí hodnot stačí a není nutné pojmenování.

4. Volání funkce a zobrazení výsledků.

```
C = statsCell(data);
disp(C)
```

   Proč: Hodnoty jsou dostupné jako `C{1}`, `C{2}`, apod.

5. Histogram známek studentů.

```
grades = randi([1,5],1,100);
histogram(grades)
xlabel('Známka'); ylabel('Počet')
```

   Proč: Histogram ukazuje rozdělení hodnot.

6. Boxplot výšky studentů.

```
heights = 160+20*randn(100,1);
boxplot(heights)
ylabel('Výška [cm]')
```

Proč: Boxplot ukazuje medián, kvartily a odlehlé hodnoty.

7. Sloupcový graf průměrů po třídách.

```
means = [2.5, 2.8, 3.1];
bar(means)
set(gca,'XTickLabel',{'Třída A','Třída B','Třída C'})
```

Proč: Bar graf je vhodný pro porovnání kategorií.

8. Výpočet a export statistik do tabulky.

```
T = table(mean(grades), std(grades), min(grades), max(grades)
    , ...
    'VariableNames',{'Mean','Std','Min','Max'});
writetable(T,'stats.csv')
```

Proč: Tabulka je snadno exportovatelná a čitelná i mimo MATLAB.

9. Uložení proměnných do MAT souboru.

```
save('studentData.mat','grades','heights')
```

Proč: MAT soubor je nativní formát MATLABu, ideální pro velké datové sady.

10. Načtení uložených dat.

```
load('studentData.mat')
```

Proč: Data lze přímo načíst zpět do workspace.

11. Výpočet korelačního koeficientu.

```
weights = 60+15*randn(100,1);
R = corrcoef(heights,weights);
```

Proč: `corrcoef` dává matici korelace, diagonála = 1, mimo-diagonální = korelace.

Matematicky: Pearsonův korelační koeficient

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

12. Interpretace korelace.

```
disp(R(1,2))
```

Proč: Hodnota blízká 1 znamená silnou pozitivní korelaci, blízká -1 negativní.

13. Scatter plot výšky vs. váhy.

```
scatter(heights,weights)
xlabel('Výška [cm]'); ylabel('Váha [kg]')
```

Proč: Vizualizace pomáhá vidět korelaci.

14. Lineární regrese pomocí polyfit.

```
p = polyfit(heights,weights,1);
yfit = polyval(p,heights);
plot(heights,weights,'o',heights,yfit,'-r')
```

Proč: Lineární regrese hledá nejlepší přímku $\hat{y} = a + bx$.

Matematicky: parametry (a, b) minimalizují součet čtverců chyb:

$$\min_{a,b} \sum (y_i - (a + bx_i))^2$$

15. Výpočet reziduí regresního modelu.

```
resid = weights - yfit;
```

Proč: Rezidua ukazují rozdíl mezi skutečnými daty a modelem.

16. Histogram reziduí.

```
histogram(resid)
xlabel('Reziduum'); ylabel('Počet')
```

Proč: Ideálně mají rezidua rozdělení blízké normálnímu.

17. Boxplot reziduí.

```
boxplot(resid)
```

Proč: Umožňuje odhalit odlehlé hodnoty a asymetrii.

18. Export výsledků regrese do tabulky.

```
Treg = table(p(1),p(2),mean(resid),std(resid),...
    'VariableNames',{'Slope','Intercept','MeanResid','
       StdResid'});
writetable(Treg,'regression.csv')
```

Proč: Uchovává parametry modelu a charakteristiky reziduí.

19. Uložení výsledků analýzy do struktury.

```
Results.corr = R(1,2);
Results.regression = p;
Results.resid = resid;
save('analysis.mat','Results')
```

Proč: Struktura drží všechny výsledky pohromadě, MAT soubor je uchovává.

20. Načtení a práce s uloženými výsledky.

```
load('analysis.mat')
disp(Results.regression)
```

Proč: Umožňuje kdykoliv pokračovat v práci bez znovuvýpočtu.

6.5 Pátek - Vnořené a anonymní funkce, práce s daty typu datetime a časovými sériemi

Cíl dne: Umět používat vnořené funkce (nested functions), anonymní funkce (@) pro rychlé výpočty, a pracovat se základními operacemi nad časovými daty.

Úkoly a řešení

1. Definice anonymní funkce pro výpočet druhé mocniny.

```
f = @(x) x.^2;
y = f(5);
```

   Proč: Anonymní funkce umožňuje rychle definovat jednoduché funkce bez zvláštního souboru.

2. Anonymní funkce pro výpočet lineární kombinace.

```
g = @(x,a,b) a*x + b;
y = g(3,2,1);  % vypočítá 2*3+1=7
```

   Proč: Parametry lze snadno předat, užitečné v regresi a fitování.

3. Použití anonymní funkce s funkcí `arrayfun`.

```
h = @(x) x.^3;
X = 1:5;
Y = arrayfun(h,X);
```

   Proč: `arrayfun` aplikuje funkci na každý prvek vektoru bez cyklu.

4. Vnořená funkce uvnitř jiné funkce.

```
function y = outerFunc(x)
    y = innerFunc(x);
    function z = innerFunc(t)
        z = t^2 + 1;
    end
end
```

   Proč: Vnořené funkce sdílí workspace s rodičovskou funkcí.

5. Vnořená funkce pro lokální výpočet součtu.

```
function s = sumOuter(x)
    s = helper(x);
    function out = helper(vals)
        out = sum(vals);
    end
end
```

   Proč: Užitečné pro organizaci kódu a skrytí pomocných funkcí.

6. Anonymní funkce jako parametr jiné funkce.

```
f = @(x) sin(x);
integral(f,0,pi)
```

Proč: Velmi užitečné v numerické matematice – např. integrace, optimalizace.

7. Vytvoření časového objektu `datetime`.

```
t = datetime(2025,9,19,14,30,0);
```

Proč: `datetime` ukládá datum a čas s vysokou přesností.

8. Aktuální čas.

```
nowTime = datetime('now');
```

Proč: Umožňuje časové razítkování dat.

9. Výpočet rozdílu dvou časů.

```
t1 = datetime(2025,9,19,8,0,0);
t2 = datetime(2025,9,19,12,30,0);
d = between(t1,t2);
```

Proč: Funkce `between` vrací rozdíl jako objekt `calendarDuration`.

10. Převod rozdílu na hodiny.

```
hoursDiff = hours(t2-t1);
```

Proč: Snadný převod na číselnou hodnotu v hodinách.

11. Generování řady časových značek.

```
times = datetime(2025,1,1):days(1):datetime(2025,1,10);
```

Proč: Ideální pro simulace časových řad (např. měření každý den).

12. Časová série s náhodnými hodnotami.

```
values = rand(1,10);
plot(times,values,'-o')
```

Proč: Čas na ose x je automaticky správně formátován.

13. Převod řetězce na `datetime`.

```
t = datetime('19-Sep-2025 14:45','InputFormat','dd-MMM-yyyy
    HH:mm');
```

Proč: Podpora různých formátů vstupních dat.

14. Výpis jednotlivých složek datumu.

```
y = year(t);
m = month(t);
d = day(t);
```

Proč: Snadná extrakce roku, měsíce, dne.

15. Přidávání časových intervalů.

```
tNew = t + hours(5) + minutes(30);
```

Proč: Praktické pro plánování a simulace.

16. Anonymní funkce pro převod minut na hodiny.

```
min2h = @(m) m/60;
result = min2h(90); % = 1.5
```

Proč: Jednoduchá převodní funkce definovaná na místě.

17. Vnořená funkce pro výpočet pracovních hodin mezi dvěma časy.

```
function h = workHours(t1,t2)
    function hh = hoursDiff(a,b)
        hh = hours(b-a);
    end
    h = hoursDiff(t1,t2);
end
```

Proč: Ukázka praktického použití vnořené funkce.

18. Týdenní průměr z denních hodnot.

```
days = datetime(2025,1,1):days(1):datetime(2025,1,30);
vals = rand(1,length(days))*10;
weekly = movmean(vals,7);
plot(days,vals,'b',days,weekly,'r','LineWidth',2)
```

Proč: `movmean` vytváří pohyblivý průměr, vhodný pro časové řady.

19. Detekce extrémních hodnot v časové sérii.

```
idx = vals > 9;
plot(days,vals,'-o'); hold on
plot(days(idx),vals(idx),'rx','MarkerSize',12)
```

Proč: Kombinace časových dat a logických podmínek pro analýzu.

20. Export časové řady do tabulky.

```
T = table(days','vals','VariableNames',{'Date','Value'});
writetable(T,'timeseries.csv')
```

Proč: Umožňuje uchovat časová data a použít je v jiných programech (Excel, Python).

6.6 Sobota – Vnořené funkce, anonymní funkce, práce s daty typu `datetime` a časové série

Cílem dne je propojit techniky práce s daty `datetime`, časovými sériemi a využitím vnořených a anonymních funkcí. Na konci vytvoříme projekt – analýzu spotřeby vody v domácnosti.

Úlohy

1. **Vytvoření vektoru časových značek pro jeden den po hodinách**

```
t = datetime(2025,9,1,0,0,0):hours(1):datetime
   (2025,9,1,23,0,0);
```

Proč: Použití konstruktoru `datetime` a operátoru `:` umožňuje snadno vytvořit časový vektor v hodinových krocích.

2. **Generování náhodné spotřeby vody (litry za hodinu)**

```
consumption = randi([20 150], 1, length(t));
```

Proč: `randi([20 150],1,n)` generuje náhodné hodnoty mezi 20–150 L. Reprezentuje realistickou spotřebu.

3. **Zobrazení spotřeby během dne (line plot)**

```
plot(t, consumption, '-o');
xlabel('Čas'); ylabel('Spotřeba [L]');
title('Spotřeba vody za den');
grid on;
```

Proč: Funkce `plot` s časovou osou automaticky formátuje datum/čas.

4. **Výpočet denního součtu spotřeby**

```
total = sum(consumption);
```

Proč: Součet všech hodinových hodnot udává celkovou denní spotřebu.

5. **Průměrná hodinová spotřeba**

```
avg = mean(consumption);
```

Proč: `mean` určí průměr. Slouží k orientačnímu porovnání jednotlivých dnů.

6. **Použití anonymní funkce pro převod L $\rightarrow$ m^3**

```
liters2m3 = @(x) x/1000;
avg_m3 = liters2m3(avg);
```

Proč: Anonymní funkce umožní jednoduchý převod bez definice samostatné funkce.

7. **Vytvoření týdenního datasetu**

```
t_week = datetime(2025,9,1,0,0,0):hours(1):datetime
   (2025,9,7,23,0,0);
cons_week = randi([20 150], 1, length(t_week));
```

Proč: Rozšíříme data na 7 dní = 168 hodin.

8. **Denní agregace pomocí `reshape`**

```
daily_matrix = reshape(cons_week, 24, []);
daily_sums = sum(daily_matrix);
```

Proč: Přeskupením do matice 24×7 snadno spočteme součty po dnech.

9. **Boxplot denní spotřeby**

```
boxplot(daily_matrix', 'Labels', 1:7);
xlabel('Den'); ylabel('Spotřeba [L]');
```

Proč: `boxplot` ukazuje rozdělení hodnot (minimum, kvartily, maximum).

10. **Vnořená funkce pro denní statistiky**

```
function stats = dailyStats(data)
    function s = oneDay(x)
        s.sum = sum(x);
        s.avg = mean(x);
    end
    stats = arrayfun(@(i) oneDay(data(:,i)), 1:size(data,2));
end
```

Proč: Vnořená funkce `oneDay` se volá uvnitř hlavní funkce a má přístup k jejím proměnným.

11. **Použití vnořené funkce**

```
stats = dailyStats(daily_matrix);
```

Proč: Funkce vrací strukturu s výsledky součtu a průměru za každý den.

12. **Maximální hodnota týdne**

```
max_val = max(cons_week);
[~, idx] = max(cons_week);
t_max = t_week(idx);
```

Proč: Najdeme hodnotu i čas, kdy nastalo maximum.

13. **Histogram týdenní spotřeby**

```
histogram(cons_week, 15);
xlabel('Spotřeba [L]'); ylabel('Četnost');
```

Proč: Histogram odhalí rozložení hodnot.

14. **Výpočet pohyblivého průměru (24h okno)**

```
mov_avg = movmean(cons_week, 24);
```

Proč: Vyhladí krátkodobé výkyvy, ukazuje trend.

15. **Porovnání spotřeby všední den vs víkend**

```
weekdays = weekday(t_week);
workdays = cons_week(weekdays <= 5);
weekends = cons_week(weekdays > 5);
mean(workdays), mean(weekends)
```

Proč: `weekday` vrací číslo dne v týdnu; lze filtrovat pracovní dny a víkend.

16. **Použití anonymní funkce pro rozdíl mezi max a min**

```
rangeFun = @(x) max(x)-min(x);
range_val = rangeFun(cons_week);
```

Proč: Anonymní funkce je praktická pro jednorázové výpočty.

17. **Rozdíly mezi po sobě jdoucími hodinami**

```
diffs = diff(cons_week);
```

Proč: `diff` ukáže, jak se měnila spotřeba mezi hodinami.

18. **Uložení datasetu do CSV a načtení zpět**

```
T = table(t_week', cons_week');
writetable(T, 'water.csv');
T2 = readtable('water.csv');
```

Proč: Export a import dat umožňuje sdílení a další zpracování.

19. **Uložení výsledků do MAT souboru**

```
save('results.mat','daily_sums','mov_avg');
```

Proč: Formát `.mat` je binární a efektivní pro ukládání více proměnných.

20. **Zobrazení trendu s anotací maxima**

```
plot(t_week, cons_week); hold on;
plot(t_max, max_val, 'ro','MarkerSize',10,'LineWidth',2);
text(t_max, max_val, ' Maximum','VerticalAlignment','bottom')
    ;
```

Proč: Anotace usnadňuje interpretaci grafu.

6.6.1 Projekt: Analýza časové série spotřeby vody

Cílem projektu je analyzovat spotřebu vody v domácnosti během 30 dní (hodinová data). Kroky: vytvoření dat, vizualizace, denní a týdenní souhrny, vnořené a anonymní funkce, export výsledků.

```
% --- Generování datasetu ---
t_all = datetime(2025,9,1,0,0,0):hours(1):datetime
   (2025,9,30,23,0,0);
cons_all = randi([20 150], 1, length(t_all));

% --- Denní agregace ---
daily_matrix = reshape(cons_all, 24, []);
daily_sums = sum(daily_matrix);
daily_means = mean(daily_matrix);

% --- Vnořená funkce pro souhrny ---
function out = dailySummary(data)
    function s = oneDay(x)
        s.sum = sum(x);
        s.avg = mean(x);
        s.max = max(x);
    end
    out = arrayfun(@(i) oneDay(data(:,i)), 1:size(data,2));
end
summaryStruct = dailySummary(daily_matrix);

% --- Anonymní funkce pro převod jednotek ---
L2m3 = @(x) x/1000;
daily_sums_m3 = L2m3(daily_sums);

% --- Pohyblivý průměr ---
mov_avg = movmean(cons_all, 24);

% --- Vizualizace ---
figure;
subplot(2,1,1);
plot(t_all, cons_all);
xlabel('Čas'); ylabel('Spotřeba [L]');
title('Spotřeba vody (30 dní)');

subplot(2,1,2);
plot(daily_sums,'-o');
xlabel('Den'); ylabel('Spotřeba [L]');
title('Denní součty');

% --- Export ---
T = table((1:30)', daily_sums', daily_means', daily_sums_m3');
T.Properties.VariableNames = {'Den','Součet_L','Prumer_L','Souč
   et_m3'};
writetable(T,'daily_water.csv');
save('water_summary.mat','T','mov_avg');
```

Proč: Projekt kombinuje práci s `datetime`, vnořenými a anonymními funkcemi, vizualizací trendů a ukládáním výsledků. Uživatel se naučí nejen zpracovat, ale i exportovat analýzu dat.

6.7 Neděle – Interaktivní analýza spotřeby energie a vody pomocí GUI a sliderů

Cílem tohoto dne je propojit všechny znalosti z předchozích týdnů a vytvořit interaktivní aplikaci v MATLABu, která umožní uživateli analyzovat data o spotřebě energie a vody. Budeme používat `uicontrol` pro vytvoření sliderů, tlačítek a menu, funkce pro vizualizaci dat (`plot`, `bar`, `histogram`) a základní statistické nástroje. Součástí je i projekt, kde budeme pracovat s daty ve formátu `datetime`, dělat pohyblivé průměry a umožníme uživateli dynamicky měnit rozsah zobrazených dat.

—

6.7.1 Ukázkový dataset: `spotreba.csv`

Soubor má 30 záznamů: sloupce `Datum`, `Energie [kWh]`, `Voda [l]`.

```
Datum,Energie,Voda
2025-01-01,12.5,140
2025-01-02,15.2,135
2025-01-03,14.8,150
2025-01-04,13.1,142
...
2025-01-30,16.4,155
```

—

6.7.2 Projekt – Interaktivní GUI aplikace

```
function spotrebaGUI()
    % Načtení dat
    data = readtable('spotreba.csv');
    cas = datetime(data.Datum,'InputFormat','yyyy-MM-dd');
    energie = data.Energie;
    voda = data.Voda;

    % Vytvoření figure
    f = figure('Name','Interaktivní analýza spotřeby',...
               'Position',[100 100 800 600]);

    % Osy grafu
    ax = axes('Parent',f,'Position',[0.1 0.3 0.8 0.65]);

    % Počáteční graf
    plot(ax,cas,energie,'-o','LineWidth',1.5);
    hold(ax,'on');
    plot(ax,cas,voda,'-s','LineWidth',1.5);
    hold(ax,'off');
    legend(ax,{'Energie [kWh]','Voda [l]'});
    xlabel(ax,'Datum'); ylabel(ax,'Spotřeba');
    title(ax,'Denní spotřeba energie a vody');
```

```
    grid(ax,'on');

    % Slider pro výběr počtu dní
    uicontrol('Style','text','String','Počet dní:',...
              'Units','normalized','Position',[0.1 0.2 0.2 0.05])
                 ;
    slider = uicontrol('Style','slider','Min',5,'Max',30,'Value'
       ,30,...
              'Units','normalized','Position',[0.3 0.2 0.5
                 0.05],...
              'Callback',@(src,~)updatePlot(round(get(src,'Value'
                 ))));

    % Tlačítko pro export
    uicontrol('Style','pushbutton','String','Exportovat do PNG'
       ,...
              'Units','normalized','Position',[0.1 0.1 0.3
                 0.07],...
              'Callback',@(~,~)exportPlot());

    % Funkce pro update grafu
    function updatePlot(N)
        plot(ax,cas(end-N+1:end),energie(end-N+1:end),'-o','
           LineWidth',1.5);
        hold(ax,'on');
        plot(ax,cas(end-N+1:end),voda(end-N+1:end),'-s','
           LineWidth',1.5);
        hold(ax,'off');
        legend(ax,{'Energie [kWh]','Voda [l]'});
        xlabel(ax,'Datum'); ylabel(ax,'Spotřeba');
        title(ax,['Spotřeba za posledních ',num2str(N),' dní']);
        grid(ax,'on');
    end

    % Funkce pro export
    function exportPlot()
        exportgraphics(ax,'spotreba_export.png','Resolution',300)
           ;
        msgbox('Graf úspěšně exportován do souboru
           spotreba_export.png');
    end
end
```

Listing 6.1: Hlavní skript pro spuštění GUI aplikace

—

6.7.3 Podrobné vysvětlení kódu

- `readtable` – načítáme data z CSV, obsahují datum a spotřebu.
- `datetime` – převádíme řetězce na časové značky.

- `uicontrol` – vytváříme interaktivní prvky GUI:
  - `slider` – posuvník určuje, kolik posledních dní zobrazit.
  - `pushbutton` – tlačítko umožňuje exportovat aktuální graf.
- `Callback` – definuje, co se stane po změně slideru nebo kliknutí na tlačítko.
- `updatePlot` – vnořená funkce dynamicky překreslí graf podle hodnoty slideru.
- `exportgraphics` – uloží obrázek v rozlišení 300 DPI do PNG.

—

6.7.4 Úlohy k procvičení

1. Upravte GUI tak, aby šlo zobrazit pouze energii nebo pouze vodu (checkboxy).
2. Přidejte výpočet průměrné denní spotřeby a zobrazte jej v textovém poli.
3. Přidejte histogram denní spotřeby energie.
4. Umožněte uživateli zvolit časový interval pomocí `ginput`.
5. Uložte exportovaný graf také jako PDF.
6. Přidejte druhý slider pro změnu hladkého pohyblivého průměru (např. 3–7 dní).
7. Zobrazte boxplot spotřeby vody.
8. Přidejte korelační graf energie vs. voda.
9. Přidejte anotaci k maximu spotřeby energie.
10. Zobrazte rozdíly mezi po sobě jdoucími dny.
11. Přidejte možnost změnit barvy čar pomocí `popupmenu`.
12. Exportujte data do Excelu pomocí `writetable`.
13. Vytvořte tlačítko pro reset GUI.
14. Přidejte časovou osu do grafu ve formátu `dd.MM`.
15. Vytvořte funkci, která počítá procentuální změny oproti předchozímu dni.
16. Zobrazte pohyblivý průměr pomocí `movmean`.
17. Přidejte druhou osu y (`yyaxis`) pro zobrazení vody a energie zvlášť.
18. Umožněte uživateli exportovat data i graf do jednoho ZIP souboru.
19. Přidejte interaktivní tooltipy pomocí `datacursormode`.
20. Vytvořte animaci, kde se graf postupně vykresluje den po dni.

6.7.5 Řešení 20 úloh v MATLAB kódu s vysvětlením

Níže jsou kompletní řešení všech 20 úloh k GUI projektu. Všechny úlohy vycházejí z principů, které jsme probírali v předchozích týdnech (grafy, práce s daty, GUI prvky, export).

—

1. **Checkboxy pro zobrazení energie/vody**

```
% Přidání dvou checkboxů
cbE = uicontrol('Style','checkbox','String','Energie',...
      'Units','normalized','Position',[0.65 0.1 0.15
         0.05],...
      'Value',1,'Callback',@(~,~)updatePlot(round(get(slider,
         'Value'))));
cbV = uicontrol('Style','checkbox','String','Voda',...
      'Units','normalized','Position',[0.8 0.1 0.15 0.05],...
      'Value',1,'Callback',@(~,~)updatePlot(round(get(slider,
         'Value'))));

% Úprava updatePlot
function updatePlot(N)
    hold(ax,'off');
    if cbE.Value
        plot(ax,cas(end-N+1:end),energie(end-N+1:end),'-o','
           LineWidth',1.5);
        hold(ax,'on');
    end
    if cbV.Value
        plot(ax,cas(end-N+1:end),voda(end-N+1:end),'-s','
           LineWidth',1.5);
    end
    hold(ax,'off');
    legend(ax,{'Energie','Voda'}); grid(ax,'on');
end
```

Proč: Checkboxy umožní uživateli interaktivně volit, zda chce zobrazit energii, vodu nebo obojí.

—

2. **Průměrná denní spotřeba**

```
% Textové pole
avgTxt = uicontrol('Style','text','Units','normalized',...
           'Position',[0.5 0.1 0.3 0.07],'String','Průměry:')
              ;

% V updatePlot doplníme výpočet
meanE = mean(energie(end-N+1:end));
meanV = mean(voda(end-N+1:end));
avgTxt.String = sprintf('E: %.2f kWh, V: %.1f l',meanE,meanV)
   ;
```

Proč: Doplňuje graf o číselnou informaci, uživatel ihned vidí průměrnou hodnotu.

—

3. **Histogram energie**

```
figure;
histogram(energie,10);
xlabel('Spotřeba energie [kWh]');
ylabel('Počet dní');
title('Histogram spotřeby energie');
```

Proč: Histogram zobrazuje rozložení spotřeby, ukazuje četnosti.

—

4. **Výběr časového intervalu pomocí ginput**

```
[x,~] = ginput(2);
tStart = datetime(x(1),'ConvertFrom','datenum');
tEnd   = datetime(x(2),'ConvertFrom','datenum');
idx = (cas>=tStart & cas<=tEnd);
plot(cas(idx),energie(idx),'-o');
```

Proč: Uživatel si může myší vybrat, jaký interval ho zajímá.

—

5. **Export také do PDF**

```
function exportPlot()
    exportgraphics(ax,'spotreba_export.png','Resolution',300)
       ;
    exportgraphics(ax,'spotreba_export.pdf');
    msgbox('Graf uložen jako PNG i PDF');
end
```

Proč: PDF formát je ideální pro publikace a LaTeX.

—

6. **Slider pro pohyblivý průměr**

```
avgSlider = uicontrol('Style','slider','Min',1,'Max',7,'Value
   ',3,...
      'Units','normalized','Position',[0.3 0.15 0.5 0.05],...
      'Callback',@(src,~)updatePlot(round(get(slider,'Value')
         )));

% V updatePlot
M = round(get(avgSlider,'Value'));
plot(ax,cas(end-N+1:end),movmean(energie(end-N+1:end),M),'-o'
   );
```

Proč: Pohyblivý průměr vyhlazuje data a umožňuje vidět trend.

—

7. **Boxplot vody**

```
figure;
boxplot(voda);
ylabel('Spotřeba vody [l]');
title('Rozptyl spotřeby vody');
```

Proč: Boxplot ukazuje medián, kvartily a extrémy.

—

8. **Korelační graf energie vs. voda**

```
figure;
scatter(energie,voda,'filled');
xlabel('Energie [kWh]'); ylabel('Voda [l]');
title('Korelace energie a vody');
corrcoef(energie,voda)
```

Proč: Zjistíme, zda spolu spotřeba vody a energie souvisí.

—

9. **Anotace maxima energie**

```
[~,idx] = max(energie);
text(cas(idx),energie(idx),'\leftarrow maximum','FontSize'
    ,12);
```

Proč: Přímá vizuální informace o maximu zvyšuje čitelnost grafu.

—

10. **Rozdíly mezi dny**

```
diffE = diff(energie);
figure; plot(cas(2:end),diffE,'-o');
xlabel('Datum'); ylabel('Rozdíl [kWh]');
title('Denní změna spotřeby energie');
```

Proč: Analýza změn může odhalit anomálie (prudký nárůst).

—

11. **Výběr barvy čar pomocí popupmenu**

```
popup = uicontrol('Style','popupmenu',...
        'String',{'Modrá','Červená','Zelená'},...
        'Units','normalized','Position',[0.7 0.15 0.2
           0.05],...
        'Callback',@(src,~)updatePlot(round(get(slider,'Value
           '))));

% Úprava updatePlot
colors = {'b','r','g'};
col = colors{popup.Value};
plot(ax,cas(end-N+1:end),energie(end-N+1:end),col,'LineWidth'
    ,1.5);
```

Proč: Uživatelské přizpůsobení zlepšuje čitelnost.

—

12. **Export dat do Excelu**

```
writetable(data,'spotreba_export.xlsx');
```

Proč: Excel je univerzální formát pro sdílení dat.

—

13. **Reset GUI**

```
uicontrol('Style','pushbutton','String','Reset',...
    'Units','normalized','Position',[0.45 0.1 0.2 0.07],...
    'Callback',@(src,~)resetGUI());

function resetGUI()
    slider.Value = 30;
    avgSlider.Value = 3;
    popup.Value = 1;
    updatePlot(30);
end
```

Proč: Reset usnadňuje návrat k výchozímu zobrazení.

—

14. **Formát časové osy**

```
ax.XTickLabel = datestr(cas(end-N+1:end),'dd.mm');
```

Proč: Česky zobrazené datum je intuitivnější.

—

15. **Procentuální změny**

```
percChange = [NaN; diff(energie)./energie(1:end-1)*100];
plot(cas,percChange,'-o');
ylabel('% změna');
```

Proč: Procenta jsou vhodná pro porovnání dynamiky.

—

16. **Pohyblivý průměr pomocí movmean**

```
figure;
plot(cas,movmean(energie,5),'-r','LineWidth',2);
```

Proč: Vyhlazování pomáhá identifikovat trendy.

—

17. **Dvojitá osa yyaxis**

```
yyaxis left
plot(cas,energie,'-o');
ylabel('Energie [kWh]');

yyaxis right
plot(cas,voda,'-s');
ylabel('Voda [l]');
```

Proč: Dvě různé jednotky lze přehledně porovnat v jednom grafu.

—

18. **Export do ZIP**

```
save('data.mat','data');
zip('export.zip',{'spotreba_export.png','data.mat'});
```

Proč: ZIP umožní pohodlné sdílení více souborů.

—

19. **Interaktivní tooltipy**

```
datacursormode on;
```

Proč: Uživatel může myší získat přesné hodnoty.

—

20. **Animace vykreslování**

```
figure;
for i=1:length(energie)
    plot(cas(1:i),energie(1:i),'-o','LineWidth',1.5);
    ylim([min(energie) max(energie)]);
    pause(0.1);
end
```

Proč: Animace umožňuje dynamickou prezentaci vývoje dat.

Kapitola 7

6. Týden Optimalizace výkonu

7.1 Pondělí: Profilování kódu v MATLABu (profile on/off, profile viewer)

Cílem dnešního bloku je naučit se měřit a analyzovat výkon kódu v MATLABu pomocí vestavěného profileru. Naučíme se:

- zapnout a vypnout profiler (`profile on/off`),
- zobrazit podrobný report v `profile viewer`,
- zjišťovat časovou náročnost jednotlivých funkcí a skriptů,
- hledat „úzká hrdla“ (bottlenecks) v kódu,
- navrhnout jednoduché optimalizace.

—

7.1.1 20 úloh s řešením a vysvětlením

1. **Zapnutí profileru před spuštěním kódu**

```
profile on
A = rand(1000);
B = A^2;
profile off
```

Proč: Příkaz `profile on` spustí měření času. Po výpočtu se profilování vypne.

—

2. **Zobrazení výsledků ve vieweru**

```
profile viewer
```

Proč: Otevře interaktivní grafické okno s podrobným rozpisem času (funkce, volání, procenta).

—

3. **Vymazání starých výsledků**

```
profile clear
```

Proč: Vyčistí předchozí záznamy, aby nové měření nebylo zkreslené.

—

4. **Profilování funkce na výpočet faktoriálu**

```
profile on
n = 500;
f = factorial(n);
profile viewer
```

Proč: Sledujeme, kolik času zabere vestavěná funkce `factorial`.

—

5. **Profilování vlastní funkce s cyklem**

```
function s = mySum(n)
    s = 0;
    for i = 1:n
        s = s + i;
    end
end

profile on
res = mySum(1e6);
profile viewer
```

Proč: Dlouhé cykly jsou často výkonově náročné; profiler ukáže jejich podíl na čase.

—

6. **Optimalizace pomocí vektorových operací**

```
profile on
s1 = mySum(1e6);          % pomalý for-cyklus
s2 = sum(1:1e6);          % rychlé vektorizované řešení
profile viewer
```

Proč: Porovnání ukazuje zásadní rozdíl mezi cykly a vektorizací.

—

7. **Profilování náhodného generátoru**

```
profile on
X = randn(10000,1000);
profile viewer
```

Proč: Ukážeme si, že generování velkých matic je časově náročná operace.

—

8. **Profilování funkce `eig`**

```
profile on
A = rand(1000);
[V,D] = eig(A);
profile viewer
```

Proč: Výpočet vlastních čísel je typicky velmi náročný, vhodný příklad pro profiler.

—

9. **Profilování vnořených funkcí**

```
function res = nestedTest(n)
    function y = inner(k)
        y = sum(1:k);
    end
    res = zeros(1,n);
    for i = 1:n
        res(i) = inner(i);
    end
end

profile on
nestedTest(1e4);
profile viewer
```

Proč: Profiler rozezná i vnořené funkce a ukáže, kde je problém.

—

10. **Profilování skriptu s více částmi**

```
profile on
part1 = rand(5000);
part2 = part1*part1;
part3 = sum(part2(:));
profile viewer
```

Proč: Vidíme rozložení výpočetního času mezi operace.

—

11. **Sledování času při práci se soubory**

```
profile on
writematrix(rand(1000),'test.csv');
M = readmatrix('test.csv');
profile viewer
```

Proč: I I/O operace mají svou časovou režii.

—

12. **Porovnání rychlosti různých metod součtu**

```
profile on
N = 1e7;
s1 = sum(1:N);
s2 = 0;
for i = 1:N
    s2 = s2 + i;
end
profile viewer
```

Proč: Ukazuje obrovský rozdíl mezi vektorizací a cyklem.

—

13. **Profilování funkce s podmínkami**

```
function y = condFun(N)
    y = zeros(1,N);
    for i=1:N
        if mod(i,2)==0
            y(i) = i^2;
        else
            y(i) = sqrt(i);
        end
    end
end

profile on
condFun(1e5);
profile viewer
```

Proč: Podmínky zpomalují kód; profiler ukáže jejich vliv.

—

14. **Sledování paměťově náročných operací**

```
profile on
big = zeros(5000);
for i=1:5000
    big(i,:) = rand(1,5000);
end
profile viewer
```

Proč: Prealokace by byla mnohem efektivnější, profiler odhalí pomalé cykly.

—

15. **Profilování funkce pro Fourierovu transformaci**

```
profile on
x = randn(1,1e6);
Y = fft(x);
profile viewer
```

Proč: Ukazuje efektivitu vestavěných algoritmů (FFT je velmi rychlá).

—

16. **Uložení výsledků profilování do struktury**

```
profile on
A = rand(3000);
B = A*A;
p = profile('info');
profile viewer
```

Proč: `profile('info')` uloží výsledky do proměnné pro další zpracování.

—

17. **Porovnání dvou algoritmů třídění**

```
x = rand(1,1e5);

profile on
sort(x); % vestavěná funkce
profile off
profile viewer
```

Proč: Vestavěné funkce jsou optimalizované, vlastní implementace by byla mnohem pomalejší.

—

18. **Profilování grafické funkce**

```
profile on
plot(rand(1,1e5));
profile viewer
```

Proč: I grafické vykreslování má časovou režii.

—

19. **Profilování funkce s rekurzí**

```
function f = recFib(n)
    if n<=2
        f=1;
    else
        f=recFib(n-1)+recFib(n-2);
    end
end

profile on
recFib(20);
profile viewer
```

Proč: Rekurze je typicky velmi pomalá, profiler to jasně ukáže.

—

20. **Profilování simulace Monte Carlo**

```
profile on
N=1e6;
x=rand(N,1);
y=rand(N,1);
inside = (x.^2 + y.^2)<=1;
pi_est = 4*sum(inside)/N;
profile viewer
```

Proč: Ukazuje kombinaci náhodného generování a logických operací, profiler odhalí největší spotřebiče času.

7.2 Úterý: Předalokace (zeros, ones, NaN) a měření výkonu (timeit)

Cílem dnešního bloku je procvičit efektivní práci s pamětí a vyhodnocování rychlosti kódu. Naučíme se:

- proč je důležitá předalokace paměti pro velké matice,
- jak používat funkce `zeros`, `ones`, `NaN`,
- jak měřit čas běhu pomocí `timeit`,
- jak porovnávat různé implementace.

—

7.2.1 20 úloh s řešením a vysvětlením

1. **Vytvoření nulové matice** 5×5

```
A = zeros(5);
```

Proč: Rychlý způsob inicializace paměti – všechny prvky jsou 0.

—

2. **Vytvoření vektoru jedniček délky 10**

```
v = ones(1,10);
```

Proč: `ones` je užitečné pro počáteční hodnoty, např. váhové vektory.

—

3. **Matice plná hodnot NaN**

```
M = NaN(3,4);
```

Proč: Hodnota NaN může signalizovat chybějící data.

—

4. **Porovnání rychlosti s a bez předalokace**

```
n = 1e5;

% Bez predalokace
tic
for i = 1:n
    a(i) = i;
end
t1 = toc;

% S predalokací
tic
b = zeros(1,n);
```

```
for i = 1:n
    b(i) = i;
end
t2 = toc;
```

Proč: Předalokace eliminuje postupné rozšiřování pole a je mnohem rychlejší.

—

5. **Použití `timeit` pro přesnější měření**

```
f = @() sum(1:1e6);
timeit(f)
```

Proč: `timeit` provede více měření a zprůměruje čas.

—

6. **Porovnání `tic/toc` a `timeit`**

```
tic
sum(1:1e6);
toc

f = @() sum(1:1e6);
timeit(f)
```

Proč: `tic/toc` měří jednorázově, zatímco `timeit` průměruje víc běhů.

—

7. **Vyplnění matice cyklem bez předalokace**

```
n = 1e4;
for i=1:n
    C(i) = i^2;
end
```

Proč: Ukázka špatné praxe – MATLAB musí stále rozšiřovat pole.

—

8. **Vyplnění matice cyklem s předalokací**

```
n = 1e4;
C = zeros(1,n);
for i=1:n
    C(i) = i^2;
end
```

Proč: Správná praxe – rychlejší a efektivnější.

—

9. **Porovnání různých předalokací**

```
n = 1e6;
A = zeros(1,n); % nulová inicializace
B = ones(1,n);  % jedničková inicializace
C = NaN(1,n);   % chybějící data
```

Proč: Různé funkce mají různé účely, ale všechny předalokují paměť.

—

10. **Použití `timeit` pro porovnání cyklu vs. vektorizace**

```
f1 = @() sum(1:1e6);
f2 = @() myLoop(1e6); % vlastní funkce s for-cyklem
time1 = timeit(f1);
time2 = timeit(f2);
```

Proč: Ukazuje, že vektorizace je dramaticky rychlejší.

—

11. **Časová náročnost tvorby velkých matic**

```
timeit(@() rand(5000));
timeit(@() zeros(5000));
```

Proč: Inicializace náhodných čísel je náročnější než vyplnění nulami.

—

12. **Vytvoření trojrozměrné matice**

```
X = zeros(100,100,50);
```

Proč: Předalokace funguje i pro 3D (např. obrazová data).

—

13. **Použití `NaN` jako markeru chybějících hodnot**

```
data = [1, 2, NaN, 4];
mean(data,'omitnan')
```

Proč: Funkce jako `mean` umí ignorovat NaN.

—

14. **Porovnání různých inicializací výkonově**

```
timeit(@() zeros(1,1e7));
timeit(@() ones(1,1e7));
timeit(@() NaN(1,1e7));
```

Proč: Všechny jsou rychlé, ale rozdíly lze měřit.

—

15. **Předalokace struktury**

```
S(1:1000) = struct('field1',0,'field2',NaN);
```

Proč: Struktury také vyžadují předalokaci pro efektivní práci.

—

16. **Měření složitější funkce**

```
f = @() det(rand(500));
timeit(f)
```

Proč: `timeit` lze použít i na těžké výpočty.

—

17. **Nesprávná předalokace – rostoucí pole**

```
x = [];
for i=1:1e5
    x = [x i];
end
```

Proč: Každým krokem se kopíruje celé pole – velmi pomalé.

—

18. **Správná předalokace pomocí zeros**

```
x = zeros(1,1e5);
for i=1:1e5
    x(i) = i;
end
```

Proč: Oproti předchozí úloze je to mnohem efektivnější.

—

19. **Test rychlosti přístupu do matice**

```
A = zeros(1000,1000);
f = @() A(500,500);
timeit(f)
```

Proč: Přístup do prvku je velmi rychlý, `timeit` to potvrdí.

—

20. **Kombinace předalokace a měření výkonu**

```
f = @() fillArray(1e5);

function x = fillArray(n)
    x = zeros(1,n);
    for i=1:n
        x(i) = i^2;
    end
end
```

```
9
10 timeit(f)
```

Proč: Praktická ukázka správného návrhu funkce s předalokací a měřením.

7.3 Středa: Vektorizace místo cyklů – přepis kódu s `for` na vektorový

Cílem dnešního bloku je osvojit si princip **vektorizace**, tj. využívání maticových a vektorových operací místo explicitních cyklů `for`. MATLAB je optimalizován pro práci s celými poli, takže vektorizace je často násobně rychlejší a přehlednější.

—

7.3.1 20 úloh s řešením a vysvětlením

1. **Součet čísel 1 až n**

```
% Původní
s = 0;
for i = 1:1000
    s = s + i;
end

% Vektorizace
s = sum(1:1000);
```

Proč: Funkce `sum` je optimalizovaná pro práci s poli.

—

2. **Umocnění všech prvků vektoru**

```
% Původní
for i=1:10
    y(i) = i^2;
end

% Vektorizace
y = (1:10).^2;
```

Proč: Použití operátoru $.umožňujeumocněnípoprvcích.$

—

3. **Součet dvou vektorů**

```
% Původní
a = 1:5; b = 10:14;
for i=1:5
    c(i) = a(i)+b(i);
end

% Vektorizace
c = a + b;
```

Proč: Operace po prvcích je vestavěná.

—

4. **Výpočet faktoriálu (n!) pomocí cyklu vs. vektorizace**

```
% Původní
n = 6; f = 1;
for i=1:n
    f = f * i;
end

% Vektorizace
f = prod(1:n);
```

Proč: Funkce `prod` násobí prvky pole.

—

5. **Výpočet skalárního součinu**

```
% Původní
a = 1:4; b = 5:8; s=0;
for i=1:4
    s = s + a(i)*b(i);
end

% Vektorizace
s = dot(a,b);
```

Proč: MATLAB má přímo funkci `dot`.

—

6. **Výpočet součtu řádků matice**

```
% Původní
M = randi(10,5,4);
for i=1:5
    r(i) = sum(M(i,:));
end

% Vektorizace
r = sum(M,2);
```

Proč: Argument 2 znamená součet po řádcích.

—

7. **Výpočet součtu sloupců matice**

```
% Původní
for j=1:4
    c(j) = sum(M(:,j));
end

% Vektorizace
c = sum(M,1);
```

—

8. **Vynásobení každého prvku konstantou**

```
% Původní
for i=1:5
    y(i) = 2*M(i);
end

% Vektorizace
y = 2*M;
```

---

9. **Výpočet odmocniny prvků pole**

```
% Původní
for i=1:10
    r(i) = sqrt(i);
end

% Vektorizace
r = sqrt(1:10);
```

---

10. **Výběr pouze sudých čísel**

```
% Původní
n=1:10; k=[];
for i=1:10
    if mod(n(i),2)==0
        k(end+1)=n(i);
    end
end

% Vektorizace
k = n(mod(n,2)==0);
```

Proč: Logické indexování.

---

11. **Nahrazení záporných prvků nulou**

```
% Původní
x = [-2 -1 0 1 2];
for i=1:length(x)
    if x(i)<0
        x(i)=0;
    end
end

% Vektorizace
x(x<0)=0;
```

---

12. Výpočet sinusů vektoru

```
% Původní
for i=1:5
    y(i) = sin(i);
end

% Vektorizace
y = sin(1:5);
```

—

13. Průměr všech prvků matice

```
% Původní
M = randi(10,3,3); s=0;
for i=1:numel(M)
    s = s + M(i);
end
avg = s/numel(M);

% Vektorizace
avg = mean(M(:));
```

—

14. Výpočet kumulativního součtu

```
% Původní
a=1:5; c=zeros(size(a));
c(1)=a(1);
for i=2:5
    c(i)=c(i-1)+a(i);
end

% Vektorizace
c = cumsum(a);
```

—

15. Výpočet rozdílů sousedních prvků

```
% Původní
a=1:5; d=zeros(1,4);
for i=1:4
    d(i)=a(i+1)-a(i);
end

% Vektorizace
d = diff(a);
```

—

16. Výběr kladných prvků

```
% Původní
x=[-3 -1 0 2 5]; y=[];
for i=1:length(x)
    if x(i)>0
        y(end+1)=x(i);
    end
end

% Vektorizace
y = x(x>0);
```

—

17. **Nahrazení hodnot větších než 10 číslem 10**

```
% Původní
x=randi(20,1,10);
for i=1:10
    if x(i)>10
        x(i)=10;
    end
end

% Vektorizace
x(x>10)=10;
```

—

18. **Výpočet absolutní hodnoty**

```
% Původní
x=-5:5; y=zeros(size(x));
for i=1:length(x)
    if x(i)<0
        y(i)=-x(i);
    else
        y(i)=x(i);
    end
end

% Vektorizace
y = abs(x);
```

—

19. **Výpočet polynomické funkce pro vektor**

```
% Původní
x=0:0.1:1;
for i=1:length(x)
    y(i)=3*x(i)^2+2*x(i)+1;
end

```

```
% Vektorizace
y = 3*x.^2+2*x+1;
```

—

20. Součin matice se skalárem

```
% Původní
M=magic(3); N=zeros(size(M));
for i=1:size(M,1)
    for j=1:size(M,2)
        N(i,j)=2*M(i,j);
    end
end

% Vektorizace
N=2*M;
```

7.4 Čtvrtek: Logické indexování místo podmínek

Cílem je ukázat, jak lze nahradit podmínky (`if`, `for`) využitím **logického indexování**. Logické indexování je v MATLABu jedním z nejefektivnějších způsobů, jak pracovat s částí dat splňující určitou podmínku, protože **odpadá explicitní iterace** a kód je kratší i rychlejší.

—

7.4.1 20 úloh s řešením a vysvětlením

1. **Výběr všech kladných prvků**

```
x = [-3 -1 0 2 5];
y = x(x > 0);
```

Proč: Podmínka `x>0` vrací logické pole stejné délky jako `x`, které vybírá jen kladné prvky.

—

2. **Nahrazení všech záporných hodnot nulou**

```
x = [-3 -1 0 2 5];
x(x < 0) = 0;
```

Proč: Logické indexy fungují i pro přiřazení.

—

3. **Výběr všech sudých čísel**

```
n = 1:10;
suda = n(mod(n,2)==0);
```

—

4. **Výběr prvků větších než průměr**

```
x = randi(20,1,10);
y = x(x > mean(x));
```

—

5. **Výběr prvků v intervalu** $\langle 5, 15 \rangle$

```
x = 1:20;
y = x(x>=5 & x<=15);
```

Proč: Operátor `&` kombinuje podmínky.

—

6. **Nahrazení prvků mimo interval nulou**

```
x = 1:20;
x(x<5 | x>15) = 0;
```

Proč: Operátor | značí logické OR.

—

7. **Výběr záporných prvků matice**

```
A = randi([-5,5],4,4);
neg = A(A<0);
```

—

8. **Nastavení všech záporných prvků matice na jejich absolutní hodnotu**

```
A(A<0) = -A(A<0);
```

—

9. **Výběr řádků matice s kladným prvním prvkem**

```
A = randi([-5,5],5,3);
rows = A(A(:,1)>0,:);
```

—

10. **Nahrazení všech NaN nulami**

```
x = [1 NaN 2 NaN 3];
x(isnan(x)) = 0;
```

—

11. **Výběr všech hodnot větších než medián**

```
x = randi(100,1,10);
y = x(x > median(x));
```

—

12. **Zvýšení všech hodnot menších než 10 o 5**

```
x = randi(20,1,10);
x(x<10) = x(x<10)+5;
```

—

13. **Výběr řádků matice, kde je v druhém sloupci hodnota > 5**

```
A = randi(10,6,3);
rows = A(A(:,2)>5,:);
```

—

14. **Výběr prvků s absolutní hodnotou menší než 3**

```
x = -5:5;
y = x(abs(x)<3);
```

---

15. **Nastavení všech hodnot větších než 100 na 100**

```
x = randi(200,1,15);
x(x>100) = 100;
```

---

16. **Výběr všech prvků na sudých indexech**

```
x = 1:10;
y = x(mod(1:length(x),2)==0);
```

---

17. **Nahrazení lichých prvků nulou**

```
x = 1:10;
x(mod(x,2)==1) = 0;
```

---

18. **Výběr prvků podle více podmínek (kladné a sudé)**

```
x = -5:5;
y = x((x>0) & (mod(x,2)==0));
```

---

19. **Výběr studentů s průměrem nad 2.0 (tabulka)**

```
T = table(["A";"B";"C"], [1.5;2.1;1.8], ...
    'VariableNames',{'Jmeno','Prumer'});
good = T(T.Prumer<2,:);
```

---

20. **Nahrazení hodnot vektorů větších než 1 směrodatná odchylka nulou**

```
x = randn(1,10);
x(abs(x)>std(x)) = 0;
```

7.5 Pátek: Srovnání vektorového a cyklického kódu – časové měření

Cílem je porovnat, jaký je rozdíl mezi **cyklickým přístupem** (pomocí `for`) a **vektorovým přístupem** (využití vektorových operací MATLABu). MATLAB je optimalizován pro práci s vektory a maticemi, proto vektorový kód bývá výrazně rychlejší. K měření výkonu použijeme `tic/toc` nebo `timeit`.

—

7.5.1 20 úloh s řešením a vysvětlením

1. **Součet čtverců prvků vektoru**

```
n = 1e6;
x = rand(1,n);

tic
s1 = 0;
for i = 1:n
    s1 = s1 + x(i)^2;
end
toc

tic
s2 = sum(x.^2);
toc
```

Proč: Vektorový kód `sum(x.^2)` je kratší a rychlejší než cyklus.

—

2. **Výpočet součinu dvou vektorů (skalární součin)**

```
a = rand(1,n);
b = rand(1,n);

tic
s1 = 0;
for i=1:n
    s1 = s1 + a(i)*b(i);
end
toc

tic
s2 = dot(a,b);
toc
```

—

3. **Sečtení prvků matice po sloupcích**

```
A = rand(1000,1000);

```

```
tic
s1 = zeros(1,1000);
for j=1:1000
    for i=1:1000
        s1(j) = s1(j) + A(i,j);
    end
end
toc

tic
s2 = sum(A,1);
toc
```

—

4. Maximum vektoru

```
tic
m1 = -Inf;
for i=1:n
    if x(i) > m1
        m1 = x(i);
    end
end
toc

tic
m2 = max(x);
toc
```

—

5. Normalizace vektoru

```
tic
y1 = zeros(1,n);
for i=1:n
    y1(i) = (x(i)-mean(x))/std(x);
end
toc

tic
y2 = (x-mean(x))/std(x);
toc
```

—

6. Součet diagonálních prvků matice

```
tic
s1 = 0;
for i=1:length(A)
    s1 = s1 + A(i,i);
```

```
end
toc

tic
s2 = trace(A);
toc
```

—

7. Výpočet faktoriálu pomocí cyklu vs. zabudované funkce

```
n = 500;

tic
f = 1;
for i=1:n
    f = f*i;
end
toc

tic
f2 = factorial(n);
toc
```

—

8. Výpočet vzdálenosti dvou vektorů

```
tic
d1 = 0;
for i=1:n
    d1 = d1 + (a(i)-b(i))^2;
end
d1 = sqrt(d1);
toc

tic
d2 = norm(a-b);
toc
```

—

9. Průměr každého řádku matice

```
tic
mean1 = zeros(size(A,1),1);
for i=1:size(A,1)
    mean1(i) = sum(A(i,:))/size(A,2);
end
toc

tic
mean2 = mean(A,2);
toc
```

---

10. Element-wise násobení dvou vektorů

```
tic
c1 = zeros(1,n);
for i=1:n
    c1(i) = a(i)*b(i);
end
toc

tic
c2 = a.*b;
toc
```

---

11. Výpočet kumulativního součtu

```
tic
c1 = zeros(1,n);
c1(1) = x(1);
for i=2:n
    c1(i) = c1(i-1)+x(i);
end
toc

tic
c2 = cumsum(x);
toc
```

---

12. Výběr sudých prvků vektoru

```
tic
evens1 = [];
for i=1:n
    if mod(x(i),2)==0
        evens1(end+1)=x(i);
    end
end
toc

tic
evens2 = x(mod(x,2)==0);
toc
```

---

13. Součet absolutních hodnot

```
tic
s1 = 0;
```

```
for i=1:n
    s1 = s1 + abs(x(i));
end
toc

tic
s2 = sum(abs(x));
toc
```

—

14. Výpočet mocnin prvků

```
tic
p1 = zeros(1,n);
for i=1:n
    p1(i) = x(i)^3;
end
toc

tic
p2 = x.^3;
toc
```

—

15. Logaritmus prvků vektoru

```
tic
l1 = zeros(1,n);
for i=1:n
    l1(i) = log(x(i));
end
toc

tic
l2 = log(x);
toc
```

—

16. Výpočet sinusu všech prvků

```
tic
s1 = zeros(1,n);
for i=1:n
    s1(i) = sin(x(i));
end
toc

tic
s2 = sin(x);
toc
```

---

17. Průměrné rozdíly mezi sousedními prvky

```
tic
diff1 = zeros(1,n-1);
for i=1:n-1
    diff1(i) = x(i+1)-x(i);
end
toc

tic
diff2 = diff(x);
toc
```

---

18. Výpočet exponenciály všech prvků

```
tic
exp1 = zeros(1,n);
for i=1:n
    exp1(i) = exp(x(i));
end
toc

tic
exp2 = exp(x);
toc
```

---

19. Spočtení počtu kladných hodnot

```
tic
count1 = 0;
for i=1:n
    if x(i)>0
        count1 = count1 + 1;
    end
end
toc

tic
count2 = sum(x>0);
toc
```

---

20. Průměrná hodnota všech druhých mocnin

```
tic
sum1 = 0;
for i=1:n
```

```
    sum1 = sum1 + x(i)^2;
end
mean1 = sum1/n;
toc

tic
mean2 = mean(x.^2);
toc
```

Shrnutí

Ve všech případech je vektorový kód:

- výrazně kratší a čitelnější,
- rychlejší díky interní optimalizaci MATLABu (využití knihoven BLAS/LAPACK),
- snadněji udržovatelný a méně náchylný k chybám.

7.6 Sobota – Projekt: Přepište pomalý skript pro výpočet součtu sousedních prvků na plně vektorový

Zadání

Máme velký vektor s milionem prvků. Původní (pomalý) skript používá cyklus `for`, aby spočítal součet sousedních prvků:

$$y_i = x_i + x_{i+1}, \quad i = 1, 2, \ldots, n-1.$$

Vaším úkolem je:

1. Implementovat tuto úlohu pomocí klasického `for`-cyklu.
2. Přepsat kód do plně vektorového tvaru bez smyček.
3. Změřit výkon obou metod (`tic`/`toc`, `timeit`).
4. Vizualizovat rozdíl v čase výpočtu.

—

7.6.1 Řešení s MATLAB kódem a podrobným vysvětlením

```
%% Projekt: Součet sousedních prvků
n = 1e6;                    % počet prvků (velký vektor)
x = rand(1,n);              % náhodný vektor

%% 1) Pomalý cyklický kód
tic
y_loop = zeros(1,n-1);      % předalokace pro efektivitu
for i = 1:n-1
    y_loop(i) = x(i) + x(i+1);
end
t_loop = toc;

%% 2) Vektorový kód
tic
y_vec = x(1:end-1) + x(2:end);
t_vec = toc;

%% 3) Ověření, že výsledky jsou stejné
disp(norm(y_loop - y_vec));  % mělo by vyjít 0 (resp. numericky
   ~1e-15)

%% 4) Porovnání rychlostí
fprintf('Čas cyklus: %.4f s\n', t_loop);
fprintf('Čas vektorový: %.4f s\n', t_vec);

%% 5) Vizualizace rozdílu v čase
figure;
bar([t_loop, t_vec])
```

```
set(gca,'XTickLabel',{'for-cyklus','vektorový'})
ylabel('Čas výpočtu [s]')
title('Porovnání výkonu: cyklus vs. vektorový kód')
```

—

7.6.2 Vysvětlení krok za krokem

1. **Generování dat:** Vytvoříme vektor `x` délky 10^6, abychom viděli rozdíly ve výkonu.
2. **Cyklický přístup:** Použijeme smyčku `for`, která iteruje $n-1$ kroků a ukládá součty sousedních prvků. Předalokace `y_loop` je nutná, jinak by MATLAB musel pole zvětšovat v každém kroku (velmi pomalé).
3. **Vektorový přístup:** Vyjádříme součet sousedních prvků přímo:

$$y = x(1 : end - 1) + x(2 : end).$$

   Tento zápis využívá tzv. **indexový posun**, tedy dvě „podmnožiny“ vektoru, které jsou o jeden prvek posunuté.
4. **Porovnání výsledků:** Funkce `norm` ukazuje, zda jsou oba vektory stejné. Rozdíl je na úrovni numerické chyby (řádově 10^{-15}).
5. **Měření času:** Funkce `tic`/`toc` změří délku výpočtu obou přístupů.
6. **Vizualizace:** Pomocí `bar` vytvoříme sloupcový graf, který porovná dobu běhu.

—

7.6.3 Proč je vektorový kód rychlejší?

- MATLAB je optimalizován na práci s celými bloky dat (vektory a matice).
- Cykly v MATLABu mají režijní náklady při každé iteraci.
- Vektorové operace jsou vnitřně implementovány v optimalizovaných knihovnách (BLAS, LAPACK).
- Výsledkem je kód, který je **kratší, čitelnější a několikanásobně rychlejší**.

7.7 Neděle – Porovnání 3 verzí algoritmu (for, vektorizace, GPU)

Zadání

Implementujte výpočet součtu sousedních prvků ve třech verzích:

$$y_i = x_i + x_{i+1}, \quad i = 1, \ldots, n-1$$

kde x je vektor délky $n = 10^7$.

Verze:

1. **For-cyklus**: klasická implementace pomocí smyčky.
2. **Vektorizace**: využití indexování `x(1:end-1) + x(2:end)`.
3. **GPU**: využití `gpuArray` a provedení výpočtu na grafické kartě.

—

MATLAB kód

```
%% Parametry
n = 1e7;
x = rand(1,n);

%% 1) For-cyklus
tic
y_for = zeros(1,n-1);
for i = 1:n-1
    y_for(i) = x(i) + x(i+1);
end
t_for = toc;

%% 2) Vektorizace
tic
y_vec = x(1:end-1) + x(2:end);
t_vec = toc;

%% 3) GPU výpočet
x_gpu = gpuArray(x);          % presun na GPU
tic
y_gpu = x_gpu(1:end-1) + x_gpu(2:end);
wait(gpuDevice);              % čekání na dokončení výpočtu
t_gpu = toc;
y_gpu = gather(y_gpu);        % zpět do CPU
%% Porovnání výsledků
fprintf('For-cyklus: %.3f s\n', t_for);
fprintf('Vektorizace: %.3f s\n', t_vec);
fprintf('GPU výpočet: %.3f s\n', t_gpu);

%% Vizualizace
```

```
figure;
bar([t_for t_vec t_gpu])
set(gca,'XTickLabel',{'for','vektorizace','GPU'})
ylabel('Čas výpočtu [s]')
title('Porovnání 3 přístupů')
```

—

Vysvětlení

1. **For-cyklus** – každý krok je interpretován MATLABem, proto trvá déle.
2. **Vektorizace** – využívá optimalizované knihovny (BLAS), dramaticky rychlejší.
3. **GPU** – přesune výpočet na grafickou kartu. Výhodné zejména pro velká data, kde overhead přesunu dat nevadí.

—

7.7.1 Test – 20 otázek (souhrn týdne 1–6)

Každá otázka má 4 možnosti (A–D). Může být správně 1–4 odpovědí. Správné odpovědi jsou uvedeny za otázkou s vysvětlením.

—

1. Co provede příkaz `A = reshape(1:9,3,3);` ?

   ```
   A) Vytvoří 3x3 jednotkovou matici
   B) Vytvoří 3x3 náhodnou matici
   C) Vytvoří matici 3x3 s čísly 1..9 po sloupcích
   D) Chyba syntaxe
   ```

   Správně: C. Reshape plní po sloupcích.

2. Co znamená příkaz `eye(4)` ?

   ```
   A) Matice všech jedniček 4x4
   B) Jednotková matice 4x4
   C) Diagonální matice s hodnotou 4
   D) Chyba
   ```

   Správně: B. `eye(n)` = jednotková matice I_n.

3. Co udělá `A .* B` ?

   ```
   A) Maticové násobení
   B) Po-prvkové násobení
   C) Transpozice A a B
   D) Součet A a B
   ```

   Správně: B. Tečka znamená po-prvkovou operaci.

4. K čemu slouží `varargin` ve funkci?

   ```
   A) Umožňuje proměnný počet vstupů
   B) Umožňuje proměnný počet výstupů
   C) Nahrazuje nargin
   D) Umožňuje anonymní funkce
   ```

   Správně: A.

5. Jaký je rozdíl mezi skriptem a funkcí?

   ```
   A) Funkce má svůj workspace
   B) Skript nemá vstupy a výstupy
   C) Funkce může vracet výstupy
   D) Skript je vždy rychlejší
   ```

   Správně: A,B,C.

6. Co udělá `plot3(x,y,z)` ?

   ```
   A) Vykreslí 2D křivku
   B) Vykreslí 3D křivku
   C) Vykreslí mřížku
   D) Slouží pro 3D scatter
   ```

   Správně: B.

7. Který příkaz uloží graf do PDF?

   ```
   A) saveas(gcf,'graf.pdf')
   B) exportgraphics(gcf,'graf.pdf')
   C) print('graf','-dpdf')
   D) writetable(T,'graf.pdf')
   ```

   Správně: A,B,C.

8. Jak se přidá popisek osy x?

   ```
   A) xlabel('čas [s]')
   B) ylabel('čas [s]')
   C) zlabel('čas [s]')
   D) title('čas [s]')
   ```

   Správně: A.

9. Jak vygenerovat náhodnou matici 3×3?

   ```
   A) rand(3)
   B) randn(3)
   C) randi(3)
   D) rand(3,3)
   ```

Správně: A,D.

10. Který příkaz načte CSV soubor do tabulky?

```
A) readmatrix('data.csv')
B) readtable('data.csv')
C) load('data.csv')
D) importdata('data.csv')
```

Správně: A,B,D.

11. Který příkaz spočítá průměr celé matice?

```
A) mean(A)
B) mean(A,'all')
C) mean2(A)
D) average(A)
```

Správně: B,C.

12. Jak vybrat řádek 2 a sloupce 1:3 z matice A?

```
A) A(2,1:3)
B) A(1:3,2)
C) A(2,:)
D) A(:,2)
```

Správně: A.

13. Který příkaz měří čas běhu?

```
A) timer
B) tic ... toc
C) profile on/off
D) timeit(@funkce)
```

Správně: B,C,D.

14. Co provede příkaz `zeros(4,5)` ?

```
A) Vytvoří 4x5 nulovou matici
B) Vytvoří 5x4 nulovou matici
C) Vytvoří 20 nul
D) Vytvoří jednotkovou matici
```

Správně: A.

15. Co znamená `x(x>0)` ?

```
A) Vrátí všechny prvky > 0
B) Vrátí všechny indexy > 0
C) Nastaví hodnoty > 0 na nulu
D) Je chyba
```

Správně: A.

16. Jak vytvořit anonymní funkci $f(x) = x^2$?

```
A) f = @(x) x^2;
B) f = x^2;
C) f = function(x) x^2;
D) f = @(x) x*x;
```

Správně: A,D.

17. Jak vytvořit 3D mřížku pro surf?

```
A) [X,Y] = meshgrid(x,y);
B) [X,Y] = grid(x,y);
C) mesh(x,y,z)
D) grid on
```

Správně: A.

18. Co udělá `save('data.mat','A')` ?

```
A) Uloží proměnnou A do souboru data.mat
B) Uloží všechny proměnné
C) Uloží CSV soubor
D) Chyba
```

Správně: A.

19. Co udělá `profile viewer` ?

```
A) Spustí měření času
B) Ukáže report výkonu
C) Ukáže obrázek
D) Zastaví skript
```

Správně: B.

20. Jaký je hlavní důvod vektorizace?

```
A) Kód je kratší
B) Kód je rychlejší
C) Kód je čitelnější
D) Je to nutné, jinak MATLAB spadne
```

Správně: A,B,C.

7.7.2 Neděle – Recapitulace: Porovnání výkonu (for vs. vektorizace vs. GPU) + Souhrnný test

7.7.3 Úvod

Cílem tohoto dne je shrnout všechny klíčové principy optimalizace výkonu v MATLABu:

- Porovnání implementací pomocí cyklu `for`, vektorizace a výpočtu na GPU,
- Měření času pomocí `tic/toc` a `timeit`,
- Zhodnocení efektivity paměti a rychlosti,
- Opakování znalostí formou testu.

—

7.7.4 1. Projekt: Porovnání 3 verzí téhož algoritmu

Cílem je spočítat vektor $y_i = \sin(x_i) + x_i^2$ pro $i = 1 \dots N$ pomocí tří metod:

1. klasický `for`-cyklus,
2. vektorizace,
3. GPU akcelerace.

—

Kód projektu

```
N = 1e7;
x = linspace(0,10,N);

% 1) Klasický for-cyklus
tic
y1 = zeros(size(x));
for i = 1:N
    y1(i) = sin(x(i)) + x(i)^2;
end
t_for = toc;

% 2) Vektorizovaná verze
tic
y2 = sin(x) + x.^2;
t_vec = toc;

% 3) GPU výpočet
if gpuDeviceCount > 0
    xg = gpuArray(x);
    tic
    yg = sin(xg) + xg.^2;
```

```
    y3 = gather(yg);
    t_gpu = toc;
else
    t_gpu = NaN;
end

% Porovnání výsledků
fprintf('Čas FOR: %.3f s\n', t_for);
fprintf('Čas Vektorizace: %.3f s\n', t_vec);
fprintf('Čas GPU: %.3f s\n', t_gpu);
```

Vysvětlení:

- `for` je nejpomalejší, neboť iteruje prvek po prvku.
- Vektorizace využívá interní C-optimalizované operace MATLABu.
- GPU akcelerace přesune výpočet na grafickou kartu (pokud dostupná).
- Funkce `gather` vrací data zpět z GPU do CPU paměti.

—

Grafické srovnání výkonu

```
bar([t_for, t_vec, t_gpu]);
set(gca,'XTickLabel',{'FOR','Vektorizace','GPU'});
ylabel('Doba výpočtu [s]');
title('Srovnání výpočetního výkonu');
```

Proč: Bar graf přehledně ukazuje úsporu času. Vektorizace bývá typicky 10–100× rychlejší než `for`, GPU může být i 500× (záleží na datové velikosti).

—

7.7.5 2. Souhrnný test – Optimalizace, datové struktury a grafy

Níže je 20 otázek, které shrnují celé 4 týdny MATLAB kurzu. Za každou otázkou je uvedeno řešení a vysvětlení.

1. **Který příkaz měří dobu běhu kódu?**

   ```
   A) clock
   B) tic/toc
   C) timer
   D) delay
   ```

   Správně: B **Proč:** `tic` spustí měření, `toc` ho ukončí a vrátí čas.

2. **Jak nejlépe inicializovat velkou matici 1000×1000?**

```
A) A = [];
B) A = zeros(1000);
C) A = rand(1000);
D) A = NaN(1000);
```

Správně: B nebo D **Proč:** Prealokace pomocí `zeros`/`NaN` zrychluje výkon.

3. **Jaká operace je nejrychlejší pro součet polí?**

```
A) for i=1:N; s=s+x(i); end
B) sum(x)
C) while loop
D) parfor loop
```

Správně: B **Proč:** `sum` je vektorizovaná C-funkce.

4. **Jak zapnete profilování?**

```
A) profiler on
B) profile on
C) tic
D) analyze on
```

Správně: B **Proč:** `profile on/off` aktivuje měření výkonnosti.

5. **Jak spojit vektory do matice?**

```
A) horzcat
B) vertcat
C) both A and B
D) append
```

Správně: C **Proč:** `horzcat` spojuje po sloupcích, `vertcat` po řádcích.

6. **Jak přidat nový prvek do struktury?**

```
S(1).name = 'test';
S(1).value = 42;
```

Proč: Nová pole lze dynamicky přidávat pomocí tečky.

7. **Jak převést tabulku na struct?**

```
A) struct2table
B) table2struct
C) cell2struct
D) cast(table)
```

Správně: B **Proč:** Funkce `table2struct` převádí každý řádek na záznam struktury.

8. **Jak vytvořit anonymní funkci?**

```
A) @ (x) x^2 + 1
B) function f(x) = x^2 + 1
C) f = inline(x^2 + 1)
D) f = @(x) x^2 + 1
```

Správně: D **Proč:** Operátor `@` definuje anonymní funkci.

9. **Který příkaz zapíše tabulku do CSV?**

```
A) save('data.csv')
B) writetable(T,'data.csv')
C) write(T)
D) fprintf(T)
```

Správně: B **Proč:** `writetable` zachová hlavičky i datové typy.

10. **Jak zjistit velikost proměnné v paměti?**

```
A) size(A)
B) memory(A)
C) whos A
D) inspect(A)
```

Správně: C **Proč:** `whos` vypíše i velikost v bajtech.

11. **Jak převést cell array na matici?**

```
A) cell2mat(C)
B) struct2cell(C)
C) mat2cell(C)
D) arrayfun(C)
```

Správně: A **Proč:** `cell2mat` spojuje numerická data do pole.

12. **Jak vynutit, aby funkce měla 2 až 4 argumenty?**

```
narginchk(2,4);
```

Proč: Zabrání chybě při nesprávném počtu vstupů.

13. **Jak validovat, že matice neobsahuje nuly?**

```
validateattributes(A,{'numeric'},{'nonzero'});
```

Proč: `validateattributes` kontroluje vlastnosti vstupů.

14. **Jak načíst všechny .csv soubory v adresáři?**

```
files = dir('*.csv');
```

Proč: Vrátí strukturu s informacemi o všech souborech.

15. **Jak získat unikátní prvky vektoru?**

    ```
    A) sort
    B) unique
    C) diff
    D) distinct
    ```

    Správně: B **Proč:** `unique` odstraní duplicity.

16. **Jak zapnout mřížku v grafu?**

    ```
    grid on
    ```

    Proč: Zlepšuje čitelnost grafů.

17. **Jak přidat popisky os s LaTeX formátováním?**

    ```
    xlabel('$x^2$','Interpreter','latex');
    ```

    Proč: Umožňuje použít matematický zápis v grafech.

18. **Jak vytvořit subplot 2×2?**

    ```
    subplot(2,2,1);
    ```

    Proč: Rozdělí okno na 4 části pro více grafů.

19. **Jak uložit proměnné do MAT souboru?**

    ```
    save('vysledky.mat','A','B');
    ```

    Proč: Binární MAT formát je efektivní pro MATLAB data.

20. **Jak načíst data zpět z MAT souboru?**

    ```
    load('vysledky.mat');
    ```

    Proč: Importuje všechny proměnné zpět do workspace.

—

Závěr

Porovnáním tří implementací (for, vektorizace, GPU) jsme demonstrovali význam optimalizace výkonu. Test poté shrnuje všechny dovednosti — od práce s maticemi a strukturami až po vizualizaci a validaci vstupů.

Kapitola 8

7. Týden Strukturovaná data

8.1 Úvod: Co jsou to strukturovaná data

V numerických i inženýrských úlohách často potřebujeme ukládat nejen číselné hodnoty, ale i texty, logické příznaky či celé kolekce dat. MATLAB proto nabízí několik typů datových struktur:

- `struct` – struktury s pojmenovanými poli,
- `cell` – buněčná pole pro heterogenní data,
- `table` – tabulky podobné databázím,
- `timetable` – časové řady.

V tomto týdnu začneme **strukturami**, které jsou základem pro organizaci složitějších dat v MATLABu.

8.2 Pondělí – Struktury: vnořené `struct` a dynamické názvy polí

Struktura je datový typ, kde jednotlivé prvky mají pojmenovaná **pole** (angl. fields). Přístup probíhá tečkovou notací (`struktura.pole`). Struktury mohou být:

- jednoduché (jednoúrovňové),
- vnořené (pole obsahuje další `struct`),
- pole struktur (více záznamů),
- dynamické (pole určujeme za běhu programu pomocí proměnné).

8.2.1 Příklady

1. **Jednoduchá struktura**

   ```
   auto.znacka = 'Škoda';
   auto.model = 'Octavia';
   auto.rok = 2018;
   ```

   Tímto vytvoříme strukturu s poli `znacka`, `model`, `rok`. —

2. **Přístup k poli**

   ```
   disp(auto.model)
   ```

   Vypíše `Octavia`. Tečková notace umožňuje přímý přístup k hodnotám. —

3. **Přidání nového pole**

   ```
   auto.barva = 'červená';
   ```

Pole lze přidávat dynamicky. —

4. **Vnořená struktura**

```
auto.motor.objem = 1.6;
auto.motor.vykon = 81;   % kW
```

Pole `motor` je samo strukturou. —

5. **Pole struktur**

```
auta(1).znacka = 'Škoda';
auta(2).znacka = 'VW';
```

Slouží k uchování více záznamů. —

6. **Prázdná struktura**

```
osoba = struct();
```

Inicializuje prázdnou strukturu. —

7. **Inicializace více polí najednou**

```
student = struct('jmeno','Eva','vek',20,'obor','Matematika');
```

Výhodné pro rychlé vytvoření. —

8. **Vnořená struktura – adresa**

```
student.adresa = struct('ulice','Hlavní','mesto','Brno');
```

Užitečné při hierarchických datech. —

9. **Pole čísel ve struktuře**

```
student.znamky = [1 2 2 3 1];
```

Struktura může obsahovat i pole. —

10. **Přístup k vnořenému poli**

```
disp(student.adresa.mesto)
```

Vrací hodnotu z vnořené struktury. —

11. **Pole struktur – více studentů**

```
studenti(1).jmeno = 'Petr';
studenti(2).jmeno = 'Jana';
studenti(3).jmeno = 'Lucie';
```

Slouží jako jednoduchá databáze. —

12. **Výpis všech jmen**

```
{studenti.jmeno}
```

Pomocí složených závorek vytvoříme cell array se jmény. —

13. **Dynamické názvy polí – zápis**

```
p = 'vek';
student.(p) = 21;
```

Umožňuje programové řízení názvů polí. —

14. **Dynamické názvy polí – čtení**

```
disp(student.(p))
```

Vrací hodnotu `21`. —

15. **Struktura v buňce**

```
C{1} = student;
```

Kombinace `cell` a `struct`. —

16. **Test na typ**

```
isstruct(student)
```

Vrací `true`, protože `student` je struktura. —

17. **Seznam polí ve struktuře**

```
fields(student)
```

Vypíše všechna pole struktury. —

18. **Odstranění pole**

```
student = rmfield(student,'obor');
```

Pole lze mazat funkcí `rmfield`. —

19. **Automatická tvorba polí ve smyčce**

```
for k=1:3
    data.(sprintf('prvek%d',k)) = rand();
end
```

Každý průchod vytvoří nové pole `prvek1`, `prvek2`, … —

20. **Převod struktury na tabulku**

```
T = struct2table(studenti);
```

Tabulka je často výhodnější pro další analýzu.

8.3 Úterý – Cell arrays: přístup k prvkům, vnořené cell

8.3.1 Úvod: Co jsou cell arrays

Cell arrays jsou speciální datová struktura MATLABu, která umožňuje ukládat heterogenní prvky, tj. kombinaci čísel, textů, logických hodnot, struktur, dalších cell, atd. Na rozdíl od běžné matice (která má všechny prvky stejného typu a dimenze), cell array umožňuje ukládat prvky různého typu i velikosti.

Zápis: `C = {...}`. Rozdíl v přístupu:

- `C{i}` – přístup k *obsahu buňky* (hodnota).
- `C(i)` – přístup k *buňce samotné* (zůstane cell).

8.3.2 20 příkladů použití cell arrays

1. **Vytvoření jednoduchého cell array**

   ```
   C = {1, 'text', true};
   ```

   Cell obsahuje číslo, řetězec a logickou hodnotu. —

2. **Indexování pomocí složených závorek**

   ```
   x = C{1};
   ```

   Vrátí číselnou hodnotu 1. Použití `{ }` znamená obsah. —

3. **Indexování pomocí kulatých závorek**

   ```
   y = C(1);
   ```

   Vrátí 1×1 cell, nikoli číslo. To je rozdíl od `C{1}`. —

4. **Přidání nového prvku**

   ```
   C{4} = pi;
   ```

   Cell lze snadno rozšířit. —

5. **Vícedimenzionální cell**

   ```
   D = {1, 2; 'A', 'B'};
   ```

   Vytvoříme 2×2 cell. —

6. **Přístup k prvku v 2D cell**

   ```
   D{2,1}
   ```

Vrátí `'A'`. —

7. **Uložení vektoru do buňky**

   ```
   C{5} = [1 2 3 4 5];
   ```

   Cell dokáže uchovat i celé pole. —

8. **Uložení matice do buňky**

   ```
   C{6} = magic(3);
   ```

   V jedné buňce je celá matice 3×3. —

9. **Cell uvnitř cell (vnořená)**

   ```
   C{7} = {'x','y','z'};
   ```

   Vnořené struktury umožňují hierarchická data. —

10. **Přístup do vnořené cell**

    ```
    C{7}{2}
    ```

    Vrátí `'y'`. —

11. **Cell array číselných vektorů**

    ```
    V = {[1 2 3], [4 5], [6]};
    ```

    Každá buňka může mít jinou délku. —

12. **Spojení všech vektorů**

    ```
    all = [V{:}];
    ```

    Pomocí `:` expandujeme cell do běžného pole. —

13. **Převod cell na char pole**

    ```
    S = {'Matlab','Octave','Python'};
    charArray = char(S);
    ```

    Výsledkem je znaková matice. —

14. **Převod cell na string array**

    ```
    stringArray = string(S);
    ```

    Modernější varianta než char matice. —

15. **Použití funkce na všechny prvky cell**

```
nums = {1, 2, 3, 4};
out = cellfun(@(x) x^2, nums);
```

Vrátí `[1 4 9 16]`. `cellfun` aplikuje funkci. —

16. **Použití funkce na stringy v cell**

```
lengths = cellfun(@length, S);
```

Vrátí délku každého slova. —

17. **Anonymní funkce s cellfun**

```
out = cellfun(@(s) upper(s), S, 'UniformOutput', false);
```

Vrátí cell se slovy velkými písmeny. —

18. **Prázdný cell array**

```
E = cell(2,3);
```

Inicializuje 2×3 prázdný cell. —

19. **Kombinace cell a struct**

```
students = {struct('jmeno','Eva','vek',21), ...
            struct('jmeno','Petr','vek',22)};
```

Každá buňka obsahuje strukturu. —

20. **Uložení různých typů do cell**

```
mix = {123, 'ABC', [1 2 3], true, magic(4)};
```

Cell je ideální pro heterogenní kolekce.

8.4 Středa – Převod mezi cell, struct a matice

8.4.1 Úvod

V MATLABu je důležité umět převádět mezi různými typy datových struktur:

- **Cell array** – flexibilní, umí uchovat různorodá data.
- **Struct** – organizovaná data se jmény polí.
- **Matice** – základní datový typ pro numerické výpočty.

Převody mezi těmito typy jsou časté – např. při načítání dat (cell), následném uložení do struktur (struct) a numerickém zpracování (matice).

—

8.4.2 20 příkladů převodů

1. **Cell na matici čísel (homogenní data)**

   ```
   C = {1, 2, 3; 4, 5, 6};
   M = cell2mat(C);
   ```

   Proč: `cell2mat` spojí číselné prvky cell do klasické matice. —

2. **Matice na cell**

   ```
   A = magic(3);
   C = num2cell(A);
   ```

   Proč: `num2cell` rozbije matici na buňky (užitečné pro další zpracování). —

3. **Pole stringů do cell**

   ```
   S = ["Matlab","Python","C++"];
   C = cellstr(S);
   ```

   Proč: `cellstr` převádí string array na cell array znaků. —

4. **Cell array na string array**

   ```
   C = {'a','b','c'};
   S = string(C);
   ```

   Proč: Novější formát string array je praktičtější pro práci s texty. —

5. **Struct na tabulku**

   ```
   students = struct('jmeno',{'Eva','Petr'}, ...
                     'vek',{20,22});
   T = struct2table(students);
   ```

Proč: Tabulka je pohodlnější pro práci s řádkově orientovanými daty. —

6. **Tabulka zpět na struct**

   ```
   S = table2struct(T);
   ```

   Proč: `table2struct` obnoví původní pole struktur. —

7. **Pole struktur na cell polí**

   ```
   names = {students.jmeno};
   ```

   Proč: Přístup k jednomu poli všech struktur a převod na cell (výpis jmen). —

8. **Pole struktur na matici čísel**

   ```
   ages = [students.vek];
   ```

   Proč: Numerická pole lze snadno spojit do vektoru. —

9. **Vnořený struct do cell**

   ```
   person.info = struct('vek',30,'mesto','Praha');
   C = struct2cell(person.info);
   ```

   Proč: `struct2cell` uloží hodnoty všech polí do cell. —

10. **Zpětný převod cell na struct**

    ```
    fields = {'vek','mesto'};
    values = {30,'Praha'};
    info = cell2struct(values,fields,2);
    ```

    Proč: `cell2struct` umí z dvojice {pole, hodnoty} sestavit strukturu. —

11. **Více struktur → cell of struct**

    ```
    S(1) = struct('id',1,'val',10);
    S(2) = struct('id',2,'val',20);
    C = num2cell(S);
    ```

    Proč: Struktury lze převést na cell pro snadnější manipulaci. —

12. **Matice na struct**

    ```
    A = rand(3,2);
    S = struct('sloupec1',A(:,1),'sloupec2',A(:,2));
    ```

    Proč: Uložení sloupců do pojmenovaných polí. —

13. **Struct na matici (číselná pole)**

```
S(1).x = 1; S(1).y = 2;
S(2).x = 3; S(2).y = 4;
M = [[S.x]' [S.y]'];
```

Proč: Přímý převod struktur na numerickou matici. —

14. **Cell pole stringů na char mat**

```
C = {'cat','dog','rat'};
M = char(C);
```

Proč: `char` vytvoří znakové pole stejné délky (doplní mezery). —

15. **Matice čísel na struct dynamicky**

```
A = magic(3);
fields = {'sl1','sl2','sl3'};
S = cell2struct(num2cell(A,1),fields,2);
```

Proč: Dynamické pojmenování polí podle sloupců matice. —

16. **Cell obsahující různé typy na struct**

```
fields = {'jmeno','vek','student'};
values = {'Lucie',23,true};
S = cell2struct(values,fields,2);
```

Proč: Převod heterogenních dat na organizovanou strukturu. —

17. **Struct na cell polí + názvy**

```
fields = fieldnames(S);
values = struct2cell(S);
```

Proč: Rozklad struktury na seznam názvů a hodnot. —

18. **Pole stringů na cell → struct**

```
F = {'x','y','z'};
V = {10,20,30};
S = cell2struct(V,F,2);
```

Proč: Převod je vhodný pro automatické generování struktur. —

19. **Cell na tabulku**

```
C = {1,'Petr'; 2,'Eva'};
T = cell2table(C,'VariableNames',{'ID','Jmeno'});
```

Proč: Pohodlná cesta k databázím v MATLABu. —

20. **Tabulka zpět na cell**

```
C = table2cell(T);
```

Proč: Někdy je potřeba mít data opět v obecném cell array.

8.5 Čtvrtek – Práce s JSON a XML v MATLABu

8.5.1 Úvod

V moderním programování a při práci s daty jsou formáty **JSON** (JavaScript Object Notation) a **XML** (Extensible Markup Language) široce používané pro výměnu dat mezi aplikacemi. MATLAB nabízí přímou podporu:

- `jsonencode` – převod MATLAB datové struktury na JSON řetězec,
- `jsondecode` – převod JSON řetězce na MATLAB datovou strukturu (struct, cell, pole),
- `xmlread`, `xmlwrite` – práce s XML soubory pomocí DOM parseru.

—

8.5.2 20 příkladů práce s JSON a XML

1. **Struct → JSON string**

   ```
   S = struct('jmeno','Petr','vek',25);
   jsonStr = jsonencode(S);
   ```

   Proč: Rychlý export struktury do JSON.

2. **JSON string → struct**

   ```
   str = '{"jmeno":"Eva","vek":22}';
   S = jsondecode(str);
   ```

   Proč: Import dat z JSON do MATLABu.

3. **Cell array → JSON**

   ```
   C = {'Matlab','Python','C++'};
   jsonStr = jsonencode(C);
   ```

   Proč: JSON přirozeně podporuje seznamy.

4. **JSON → cell array**

   ```
   str = '["auto","vlak","kolo"]';
   C = jsondecode(str);
   ```

   Proč: Převede JSON pole na MATLAB cell.

5. **Matice čísel → JSON**

   ```
   A = [1 2 3; 4 5 6];
   jsonStr = jsonencode(A);
   ```

Proč: MATLAB matice se převede na JSON pole polí.

6. **JSON s vnořenou strukturou**

```
S = struct('student',struct('jmeno','Karel','vek',21));
jsonStr = jsonencode(S);
```

Proč: Vnořené struktury se převádějí na vnořená JSON pole.

7. **Dekódování vnořeného JSONu**

```
str = '{"student":{"jmeno":"Pavel","vek":23}}';
S = jsondecode(str);
disp(S.student.jmeno);
```

Proč: Lze snadno přistupovat k vnořeným polím.

8. **Pole struktur → JSON array**

```
S(1) = struct('id',1,'val',10);
S(2) = struct('id',2,'val',20);
jsonStr = jsonencode(S);
```

Proč: JSON reprezentuje MATLAB pole struktur jako pole objektů.

9. **JSON array → pole struktur**

```
str = '[{"id":1,"val":10},{"id":2,"val":20}]';
S = jsondecode(str);
```

Proč: MATLAB převede JSON pole objektů na pole struktur.

10. **Export JSON do souboru**

```
S = struct('mesto','Praha','obyvatele',1300000);
jsonStr = jsonencode(S);
fid = fopen('data.json','w');
fprintf(fid,'%s',jsonStr);
fclose(fid);
```

Proč: Uložení JSON na disk pro sdílení.

11. **Načtení JSON ze souboru**

```
fid = fopen('data.json','r');
raw = fread(fid,inf);
str = char(raw');
fclose(fid);
S = jsondecode(str);
```

Proč: Import JSON z externího souboru.

12. **XML zápis do souboru**

```
docNode = com.mathworks.xml.XMLUtils.createDocument('root');
root = docNode.getDocumentElement;
child = docNode.createElement('name');
child.appendChild(docNode.createTextNode('Matlab'));
root.appendChild(child);
xmlwrite('data.xml',docNode);
```

Proč: Vytvoření a uložení XML dokumentu.

13. **Čtení XML souboru**

```
doc = xmlread('data.xml');
root = doc.getDocumentElement;
nameNode = root.getFirstChild;
disp(char(nameNode.getTextContent));
```

Proč: XML lze načítat a procházet přes DOM.

14. **XML s více elementy**

```
child2 = docNode.createElement('vek');
child2.appendChild(docNode.createTextNode('25'));
root.appendChild(child2);
xmlwrite('data.xml',docNode);
```

Proč: Přidání více uzlů do XML.

15. **Iterace přes XML elementy**

```
list = root.getChildNodes;
for i = 1:list.getLength
    node = list.item(i-1);
    if node.getNodeType == node.ELEMENT_NODE
        disp(char(node.getTagName));
    end
end
```

Proč: Procházení všech uzlů v XML stromu.

16. **Konverze XML → struktura (vlastní parser)**

```
doc = xmlread('data.xml');
root = doc.getDocumentElement;
S.name = char(root.getElementsByTagName('name').item(0).getTextContent);
S.vek  = str2double(root.getElementsByTagName('vek').item(0).getTextContent);
```

Proč: XML lze převést na MATLAB struct.

17. **Uložení cell array do JSON**

```
C = {1,'text',true,[1 2 3]};
jsonStr = jsonencode(C);
```

Proč: JSON podporuje heterogenní seznamy podobně jako cell.

18. **Načtení komplexního JSON**

```
str = '{"auto":{"znacka":"Škoda","rok":2020}, ...
        "majitel":{"jmeno":"Pavel"}}';
S = jsondecode(str);
disp(S.auto.znacka);
```

Proč: MATLAB bez problému zvládne složitější hierarchii JSON.

19. **JSON → tabulka**

```
str = '[{"jmeno":"Eva","vek":20},{"jmeno":"Petr","vek":21}]';
S = jsondecode(str);
T = struct2table(S);
```

Proč: Data z JSON lze přímo převést do tabulky pro analýzu.

8.6 Pátek – Načítání dat z více souborů do jednoho `structu`

8.6.1 Úvod

V praxi často pracujeme s mnoha datovými soubory (např. CSV, MAT, TXT), které reprezentují různé experimenty nebo části měření. Abychom je mohli efektivně spravovat, je výhodné nahrát je do jedné datové struktury – typicky `struct`, kde každý prvek odpovídá jednomu souboru.

Tento postup přináší:

- Přehlednost (všechny soubory v jedné proměnné),
- Snazší zpracování (můžeme iterovat přes strukturu),
- Možnost uchovat metadata (název souboru, cesta, parametry).

—

8.6.2 20 příkladů načítání a práce se strukturami

1. **Načtení jednoho CSV souboru do structu**

```
data1 = readmatrix('soubor1.csv');
S(1).name = 'soubor1.csv';
S(1).data = data1;
```

Proč: Každý soubor má v poli struct vlastní záznam.

2. **Načtení druhého souboru**

```
data2 = readmatrix('soubor2.csv');
S(2).name = 'soubor2.csv';
S(2).data = data2;
```

Proč: Jednoduše přidáváme nové položky.

3. **Použití cyklu pro více souborů**

```
files = {'soubor1.csv','soubor2.csv','soubor3.csv'};
for i = 1:numel(files)
    S(i).name = files{i};
    S(i).data = readmatrix(files{i});
end
```

Proč: Automatizuje načítání větší sady souborů.

4. **Načítání s absolutní cestou**

```
folder = 'C:\data\';
files = {'a.csv','b.csv'};
for i = 1:numel(files)
    fname = fullfile(folder,files{i});
    S(i).data = readmatrix(fname);
end
```

Proč: `fullfile` zajišťuje správné spojení cest.

5. **Uložení metadat – čas načtení**

```
S(i).importTime = datetime('now');
```

Proč: U každého souboru uchováme čas importu.

6. **Uložení velikosti dat**

```
S(i).size = size(S(i).data);
```

Proč: Struktura může obsahovat i odvozené informace.

7. **Načtení tabulkových dat (readtable)**

```
S(i).table = readtable(files{i});
```

Proč: Pro CSV s hlavičkou je vhodnější tabulka.

8. **Načtení více MAT souborů**

```
matfiles = {'m1.mat','m2.mat'};
for i = 1:numel(matfiles)
    tmp = load(matfiles{i});
    S(i).vars = tmp;
end
```

Proč: `load` přímo importuje MATLAB proměnné.

9. **Uložení názvu proměnných v MAT souboru**

```
info = whos('-file','m1.mat');
S(1).variables = {info.name};
```

Proč: Získáme metadata o obsahu souboru.

10. **Načtení všech CSV v adresáři**

```
files = dir('*.csv');
for i = 1:numel(files)
    S(i).name = files(i).name;
    S(i).data = readmatrix(files(i).name);
end
```

Proč: Automatické zpracování všech souborů.

11. **Filtrování podle velikosti dat**

```
bigFiles = [S.size];
idx = find(bigFiles(:,1) > 100);
```

Proč: Můžeme hledat jen soubory s více než 100 řádky.

12. **Načtení JSON souborů do structu**

```
raw = fileread('student.json');
S(1).json = jsondecode(raw);
```

Proč: Struktura může obsahovat i dekódovaná data.

13. **Načtení XML souborů do structu**

```
doc = xmlread('data.xml');
S(1).xml = doc;
```

Proč: Lze kombinovat různé formáty v jednom structu.

14. **Sloučení všech dat do jedné matice**

```
allData = vertcat(S.data);
```

Proč: Snadné sloučení obsahu všech souborů.

15. **Načtení + výpočet statistiky**

```
for i = 1:numel(S)
    S(i).meanVal = mean(S(i).data(:));
end
```

Proč: Každému záznamu přidáme průměr dat.

16. **Uložení structu do MAT souboru**

```
save('allData.mat','S');
```

Proč: Celou databázi lze uložit jako jeden soubor.

17. **Načtení více souborů s proměnným počtem sloupců**

```
for i = 1:numel(files)
    T = readtable(files{i});
    S(i).headers = T.Properties.VariableNames;
    S(i).table = T;
end
```

Proč: Tabulky zvládají různé struktury dat.

18. **Výběr jednoho souboru podle jména**

```
idx = strcmp({S.name},'soubor2.csv');
disp(S(idx).data);
```

Proč: Snadno vybereme konkrétní dataset.

19. **Iterace a zpracování všech souborů**

```
for i = 1:numel(S)
    plot(S(i).data(:,1), S(i).data(:,2));
    hold on;
end
hold off;
```

Proč: Lze automaticky vykreslit data ze všech souborů.

8.7 Sobota – Projekt: Struktury, JSON a vizualizace trendů

8.7.1 Úvod

V této projektové úloze si procvičíme práci se strukturami a formáty dat. Cílem je:

- Uložit výsledky měření do `struct`,
- Exportovat data do formátu JSON (`jsonencode`),
- Načíst více CSV souborů, sjednotit je do jedné struktury,
- Vizualizovat trend měření (časová osa vs. hodnota).

Takový postup je běžný v reálných aplikacích, např. při záznamu spotřeby energie, laboratorních měřeních nebo ukládání dat senzorů.

—

8.7.2 Část 1: Uložení výsledků do struktury a export do JSON

```
% Příklad dat z měření
cas = (1:10)';                      % časové body (1 až 10 s)
napeti = rand(10,1)*5;              % náhodná napětí (0-5 V)
proud = rand(10,1)*2;               % náhodné proudy (0-2 A)

% Uložení do struktury
Mereni.cas = cas;
Mereni.napeti = napeti;
Mereni.proud = proud;
Mereni.datum = datestr(now);

% Export do JSON
jsonStr = jsonencode(Mereni);
fid = fopen('mereni.json','w');
fprintf(fid, '%s', jsonStr);
fclose(fid);
```

Proč:

- Struktura `Mereni` uchovává různé veličiny pohromadě, což je přehlednější než více samostatných proměnných.
- Funkce `jsonencode` převede strukturu do textového JSON formátu.
- Tento formát je univerzální a snadno čitelný v Pythonu, C++, JavaScriptu i MATLABu.

—

8.7.3 Část 2: Sjednocení více CSV souborů a vizualizace trendů

Předpokládejme, že máme několik souborů CSV se spotřebou energie (`spotreba1.csv`, `spotreba2.csv`, ...). Každý obsahuje dva sloupce: `čas (dny)` a `spotřeba (kWh)`.

```
% Seznam souborů
files = {'spotreba1.csv','spotreba2.csv','spotreba3.csv'};

% Inicializace struktury
for i = 1:numel(files)
    T = readtable(files{i});
    Data(i).filename = files{i};
    Data(i).cas = T{:,1};      % první sloupec = čas
    Data(i).spotreba = T{:,2}; % druhý sloupec = spotřeba
end

% Spojení všech datasetů do jednoho vektoru
allCas = vertcat(Data.cas);
allSpotreba = vertcat(Data.spotreba);

% Vizualizace trendu
plot(allCas, allSpotreba,'-o','LineWidth',2);
xlabel('Čas (dny)');
ylabel('Spotřeba [kWh]');
title('Sjednocený trend spotřeby energie');
grid on;
```

Proč:

- Každý CSV soubor reprezentuje jeden dataset – načteme ho do struktury.
- `vertcat` sjednotí jednotlivé vektory do jedné osy času a jedné osy spotřeby.
- Výsledkem je graf trendu spotřeby přes více souborů – přehledná vizualizace.

—

8.7.4 Rozšíření projektu

- Přidejte do struktury statistiky (průměr, maximum, minimum) pro každý soubor.
- Exportujte sjednocený dataset do CSV (`writetable`).
- Uložte celý `Data` struct do MAT souboru (`save`).

8.8 Neděle – Recap: Načtení JSON databáze a souhrnný test

8.8.1 Úvod

Závěrečný den 7. týdne kombinuje vše, co jsme se naučili: práci s daty, strukturami, JSON formáty i vizualizací. Vytvoříme větší databázi měření, uložíme ji jako `JSON`, načteme zpět do MATLABu a vytvoříme souhrnný graf. Na závěr následuje test shrnující celý týden o strukturovaných datech.

—

8.8.2 1. Projekt – práce s větší databází v JSON

Představme si dataset z měření teploty a vlhkosti během 10 dní na třech různých místech.

```
% Vytvoření větší databáze
Stanoviste = {'Laborator','Sklad','Exteriér'};
for i = 1:numel(Stanoviste)
    Data(i).stanoviste = Stanoviste{i};
    Data(i).cas = (datetime(2025,10,1) : days(1) : datetime(2025,10,10))';
    Data(i).teplota = 20 + randn(10,1)*2 - i;   % různé teploty
    Data(i).vlhkost = 50 + randn(10,1)*5 + i*3; % různá vlhkost
end

% Uložení databáze do JSON
jsonStr = jsonencode(Data);
fid = fopen('mereni_databaze.json','w');
fprintf(fid,'%s',jsonStr);
fclose(fid);

disp('Databáze úspěšně exportována.');
```

Proč:

- Struktura `Data` obsahuje 3 podsoubory (stanoviště).
- Každé stanoviště má vlastní měření teploty a vlhkosti v čase.
- JSON umožňuje univerzální uložení pro sdílení i další analýzu.

—

8.8.3 2. Načtení JSON zpět a vizualizace trendu

```
% Načtení JSON zpět
raw = fileread('mereni_databaze.json');
Mereni = jsondecode(raw);

% Vykreslení všech stanovišť
figure; hold on;
```

```
for i = 1:numel(Mereni)
    plot(Mereni(i).cas, Mereni(i).teplota, '-o', 'LineWidth', 1.5);
end
hold off;
xlabel('Datum');
ylabel('Teplota [°C]');
title('Trend teploty ve všech stanovištích');
legend({Mereni.stanoviste}, 'Location', 'best');
grid on;
```

8.8.4 Uložení a načtení měření z JSON souboru v MATLABu

8.8.5 1) Uložení měření do JSON souboru

```
Stanoviste = {'Laborator','Sklad','Exterier'};
```

Definuje buněčné pole se třemi textovými názvy stanovišť, která představují jednotlivá místa měření.

```
for i = 1:numel(Stanoviste)
```

Zahajuje cyklus, který projde všechna stanoviště. Funkce `numel` vrací počet prvků v poli.

```
    Data(i).stanoviste = Stanoviste{i};
```

Každému prvku struktury `Data` přiřazuje název stanoviště. Vznikne strukturované pole `Data(1)`, `Data(2)`, `Data(3)`.

```
    cas = (datetime(2025,10,1) : days(1) : datetime(2025,10,10))';
```

Vytvoří vektor datových objektů `datetime` od 1. 10. 2025 do 10. 10. 2025 s krokem jeden den.

```
    Data(i).cas = cellstr(datestr(cas, 'yyyy-mm-dd'));
```

Převede hodnoty typu `datetime` na textové řetězce (formát ISO `YYYY-MM-DD`) a uloží je do buněčného pole. JSON neumožňuje ukládat `datetime` přímo, proto se používá textová reprezentace.

```
    Data(i).teplota = 20 + randn(10,1)*2 - i;
```

Generuje náhodná data teploty. `randn(10,1)` vytvoří vektor 10 náhodných hodnot s normálním rozdělením, které jsou posunuty o 20 °C a sníženy o `i`, aby každé stanoviště mělo mírně jinou průměrnou teplotu.

```
    Data(i).vlhkost = 50 + randn(10,1)*5 + i*3;
```

Generuje náhodná data vlhkosti. Průměrná hodnota je 50 %, rozptyl 5 %, a pro vyšší index stanoviště se vlhkost zvyšuje o `i*3`.

```
end
```

Ukončení cyklu `for`.

```
jsonStr = jsonencode(Data);
```

Převede celou strukturu `Data` na text ve formátu JSON (serializace dat).

```
fid = fopen('mereni_databaze.json','w');
fprintf(fid,'%s',jsonStr);
fclose(fid);
```

Otevře soubor pro zápis, zapíše textový JSON řetězec a poté soubor uzavře.

```
disp('Databáze úspěšně exportována.');
```

Vypíše potvrzovací hlášení o úspěšném exportu.

—

8.8.6 2) Načtení dat z JSON a vizualizace teplotního trendu

```
raw = fileread('mereni_databaze.json');
```

Načte obsah souboru jako jeden textový řetězec do proměnné `raw`.

```
Mereni = jsondecode(raw);
```

Převede načtený JSON text zpět do MATLAB struktury (deserializace).

```
figure; hold on;
```

Otevře nové okno s grafem a zapne režim `hold on`, aby bylo možné vykreslit více křivek do jednoho obrázku.

```
for i = 1:numel(Mereni)
```

Zahájí cyklus přes všechna stanoviště.

```
    cas_dt = datetime(Mereni(i).cas, 'InputFormat', 'yyyy-MM-dd');
```

Převede načtené textové datumy zpět na objekt `datetime`. Parametr `InputFormat` musí odpovídat použitému formátu při exportu.

```
    plot(cas_dt, Mereni(i).teplota, '-o', 'LineWidth', 1.5);
```

Vykreslí průběh teploty v čase pro i-té stanoviště. Parametr `'-o'` určuje spojnicový graf s kruhovými značkami.

```
end
hold off;
```

Po vykreslení všech stanovišť se režim `hold` vypne.

```
xlabel('Datum');
ylabel('Teplota [°C]');
title('Trend teploty ve všech stanovištích');
legend({Mereni.stanoviste}, 'Location', 'best');
grid on;
```

Popisuje osy, přidává název grafu, automaticky generovanou legendu a zapíná mřížku.

—

8.8.7 Shrnutí

- **Převod dat:** Pro JSON export je nutné převést typ `datetime` na text.
- **Serializace:** Funkce `jsonencode` a `jsondecode` zajišťují univerzální přenos strukturovaných dat.
- **Generování dat:** Funkce `randn()` simuluje reálné kolísání měřených veličin.
- **Vizualizace:** Pomocí `plot`, `legend`, `xlabel`, `grid on` je vytvořen přehledný časový trend teplot pro všechna stanoviště.

8.8.8 Uložení a vizualizace dat o teplotě a vlhkosti ve formátu JSON

8.8.9 1) Uložení měření do JSON souboru

```
Stanoviste = {'Laborator','Sklad','Exterier'};
```

Definuje názvy tří měřicích stanovišť, uložené jako buněčné pole textových řetězců.

```
for i = 1:numel(Stanoviste)
```

Spouští cyklus, který postupně vytvoří záznamy pro všechna stanoviště.

```
    Data(i).stanoviste = Stanoviste{i};
```

Do struktury `Data(i)` uloží název aktuálního stanoviště.

```
    cas = (datetime(2025,10,1) : days(1) : datetime(2025,10,10))';
```

Vytváří vektor dat typu `datetime` s krokem jeden den. Vektor má 10 prvků (1. – 10. října 2025).

```
    Data(i).cas = cellstr(datestr(cas, 'yyyy-mm-dd'));
```

Převádí hodnoty typu `datetime` na textové řetězce formátu `YYYY-MM-DD`, aby byly kompatibilní s JSON formátem.

```
    Data(i).teplota = 20 + randn(10,1)*2 - i;
```

Simuluje teplotní data pro dané stanoviště. Funkce `randn(10,1)` generuje náhodný šum se směrodatnou odchylkou 2 a průměrem 20 °C, který je mírně snížen podle indexu stanoviště.

```
    Data(i).vlhkost = 50 + randn(10,1)*5 + i*3;
```

Simuluje hodnoty vlhkosti vzduchu, průměr 50 % s rozptylem 5 % a rostoucí střední hodnotou podle indexu stanoviště.

```
end
```

Ukončuje cyklus, čímž vznikne struktura s třemi stanovišti a jejich měřením.

```
jsonStr = jsonencode(Data);
```

Převede strukturu `Data` do textového JSON formátu.

```
fid = fopen('mereni_databaze.json','w');
fprintf(fid,'%s',jsonStr);
fclose(fid);
```

Otevře soubor `mereni_databaze.json` pro zápis, uloží JSON text a uzavře soubor.

```
disp('Databáze úspěšně exportována.');
```

Zobrazí potvrzovací zprávu o úspěšném exportu dat.

—

8.8.10 2) Načtení JSON a vizualizace teploty i vlhkosti

```
raw = fileread('mereni_databaze.json');
```

Načte obsah JSON souboru jako textový řetězec.

```
Mereni = jsondecode(raw);
```

Převede textový JSON zpět na MATLAB strukturu. Každý prvek odpovídá jednomu stanovišti.

```
figure;
subplot(2,1,1); hold on;
```

Otevře nové grafické okno a vytvoří první podgraf (1. řádek z 2). Režim `hold on` zajistí, že všechny stanoviště budou vykresleny v jednom podgrafu.

```
for i = 1:numel(Mereni)
    cas_dt = datetime(Mereni(i).cas, 'InputFormat', 'yyyy-MM-dd');
    plot(cas_dt, Mereni(i).teplota, '-o', 'LineWidth', 1.5);
end
```

Pro každé stanoviště převede textová data na typ `datetime` a vykreslí průběh teploty. Používá spojnicový graf s kruhovými značkami.

```
hold off;
xlabel('Datum');
ylabel('Teplota [°C]');
title('Trend teploty ve všech stanovištích');
legend({Mereni.stanoviste}, 'Location', 'best');
grid on;
```

Popíše osy, přidá název, legendu a zapne mřížku pro čitelnost grafu.

```
subplot(2,1,2); hold on;
```

Vytvoří druhý podgraf (2. řádek z 2), určený pro vykreslení vlhkosti.

```
for i = 1:numel(Mereni)
    cas_dt = datetime(Mereni(i).cas, 'InputFormat', 'yyyy-MM-dd');
    plot(cas_dt, Mereni(i).vlhkost, '-s', 'LineWidth', 1.5);
end
```

Opět převádí text na `datetime`, ale nyní vykresluje vlhkost. Použit je čtvercový marker `'-s'` pro odlišení od teplotního grafu.

```
hold off;
xlabel('Datum');
ylabel('Vlhkost [%]');
title('Trend vlhkosti ve všech stanovištích');
legend({Mereni.stanoviste}, 'Location', 'best');
grid on;
```

Popíše druhý graf obdobně jako první, tentokrát pro vlhkost.

—

8.8.11 Shrnutí

- **Struktura dat:** Každé stanoviště obsahuje časové údaje, teplotu a vlhkost. Data jsou uložena jako pole struktur.
- **Ukládání a přenos:** Pomocí `jsonencode` se data převedou do univerzálního formátu JSON, vhodného i pro jiné jazyky (Python, C++, web).
- **Vizualizace:** Funkce `subplot` umožňuje přehledné zobrazení více veličin v jednom okně. Oba grafy sdílejí časovou osu, což usnadňuje srovnání trendů.
- **Časová osa:** Správné převedení typu `datetime` na text a zpět je nezbytné pro kompatibilitu JSON a pro funkční grafické vykreslení.

Proč:

- JSON data jsou převedena zpět na strukturu MATLABu funkcí `jsondecode`.
- Smyčka prochází jednotlivá stanoviště a vykreslí průběh teploty.
- `legend` se automaticky generuje z názvů stanovišť.

—

8.8.12 Rozšíření

- Do struktury lze přidat i další senzory (tlak, CO_2, světlo).
- Pro reálná měření je vhodné použít časové řady typu `timetable`.
- Export do JSON je ideální pro webové aplikace či komunikaci mezi MATLABem a Pythonem.

—

8.8.13 Souhrnný test – Strukturovaná data a formáty

Každá otázka má 1–4 správné odpovědi.

1. Který příkaz načte JSON soubor do struktury?

   A) `readtable()`
   B) `loadjson()`
   C) `fileread() + jsondecode()`
   D) `importdata()`

   Správně: C *Proč:* MATLAB používá dvojici `fileread` (načtení textu) a `jsondecode` (dekódování JSON).

2. Která z funkcí uloží proměnné MATLABu do binárního formátu?

   A) `save()`
   B) `writetable()`
   C) `fprintf()`
   D) `jsonencode()`

   Správně: A *Proč:* `save` ukládá do .mat formátu.

3. Jak získáme seznam všech CSV souborů v adresáři?

   A) `dir('*.csv')`
   B) `ls *.csv`
   C) `listfiles('csv')`
   D) `filelist('csv')`

   Správně: A *Proč:* Funkce `dir` vrací strukturu se jmény souborů daného typu.

4. Jak přidáme k poli struktury nové pole?

   A) `S.data = rand(5,1);`
   B) `append(S,rand(5,1));`
   C) `S{end+1} = rand(5,1);`
   D) `S(end+1).data = rand(5,1);`

   Správně: D *Proč:* Struktury používají tečkovou notaci, indexy se rozšiřují pomocí `end+1`.

5. Co vrátí výraz `fieldnames(S)`?

   A) `Počet polí struktury`
   B) `Seznam názvů polí`
   C) `Hodnoty všech polí`
   D) `Datové typy proměnných`

   Správně: B *Proč:* `fieldnames` vypíše názvy polí struktury jako cell array.

6. K čemu slouží `jsonencode()`?

   ```
   A) Načte JSON do struktury
   B) Převádí MATLAB proměnnou na JSON text
   C) Otevře soubor JSON v editoru
   D) Kóduje pole znaků na base64
   ```

   Správně: B

7. Co udělá příkaz `vertcat(S.data)`?

   ```
   A) Sloučí všechna pole do jedné tabulky
   B) Sjednotí vertikálně datové části všech struktur
   C) Provede horizontální spojení struktur
   D) Uloží data do MAT souboru
   ```

   Správně: B

8. Jak zjistíme velikost pole `S.data`?

   ```
   A) numel(S.data)
   B) size(S.data)
   C) length(S)
   D) fieldnames(S)
   ```

   Správně: B

9. Jak uložit strukturu do JSON souboru?

   ```
   A) save('data.json','S')
   B) writetable(S,'data.json')
   C) jsonStr = jsonencode(S); fprintf(fid,jsonStr)
   D) encodejson(S)
   ```

   Správně: C

10. Jak vybereme všechny názvy polí ve struktuře S?

    ```
    A) names = getfields(S);
    B) names = fieldnames(S);
    C) names = structinfo(S);
    D) names = keys(S);
    ```

    Správně: B

11. Co znamená `S(1).data(3)`?

    ```
    A) 3. pole struktury
    B) Třetí prvek pole data ve 1. struktuře
    C) Indexace matice v datech
    D) Pole textu
    ```

Správně: B

12. K čemu slouží `saveas(gcf,'graf.png')`?

```
A) Uloží graf jako obrázek
B) Uloží proměnnou gcf do PNG
C) Otevře graf v okně
D) Konvertuje JSON na PNG
```

Správně: A

13. Jak získat všechny hodnoty teploty z pole Mereni?

```
A) [Mereni.teplota]
B) Mereni(:, 'teplota')
C) Mereni.teplota(:)
D) sum(Mereni.teplota)
```

Správně: A

14. Jak načíst MAT soubor do struktury?

```
A) readtable('data.mat')
B) importdata('data.mat')
C) load('data.mat')
D) open('data.mat')
```

Správně: C

15. Co dělá `table2struct(T)`?

```
A) Převede tabulku na strukturu
B) Uloží tabulku do JSON
C) Převede strukturu na tabulku
D) Převádí cell array na vektor
```

Správně: A

Část III

3 a 4 Měsíc - Pokročilé techniky Cíl : naučit se OOP, modularitu, GUI

Část IV

5 Měsíc - Integrace a rozšíření Cíl : propojit MATLAB s Pythonem, C/C++, API, a vytvářet distribuovatelné aplikace.

Část V

6 Měsíc - Větší projekty Cíl : vytvořit plnohodnotné aplikace s dokumentací a optimalizací.

Část VI

Přílohy

Kapitola 9

Matematika

9.1 Cauchyova úloha – teorie a implementace v MATLABu

9.1.1 Zadání

Řešíme počáteční úlohu pro obyčejnou diferenciální rovnici

$$y'(x) = \frac{4x - y(x)}{x}, \quad y(1) = 3, \quad x \in [1, 3].$$

9.1.2 Analytické řešení

Pomocí metody integrujícího faktoru (nebo symbolických výpočtů) získáme obecné řešení lineární rovnice

$$y(x) = 4x - 1 + \frac{C}{x}.$$

Dosazením počáteční podmínky $y(1) = 3$ dostaneme $C = 0$, tedy

$$y(x) = 4x - 1.$$

9.1.3 Numerické metody

- **Funkce `ode45`** – implementuje Rungeho–Kuttovu metodu 4.–5. řádu s adaptivním krokem, která je vhodná pro širokou třídu úloh.
- **Eulerova metoda** – explicitní metoda prvního řádu. Je jednoduchá, ale méně přesná, zvláště při větších krocích.

9.1.4 MATLAB kód

Následující kód ukazuje symbolické řešení a jeho porovnání s numerickým řešením pomocí `ode45` a Eulerovy metody.

```
% Uloha: Reseni Cauchyovy ulohy
% y' = (4*x - y)/x,  y(1) = 3,  interval [1,3]

% --- Symbolicke reseni ---
syms x y(x)
ode = diff(y,x) == (4*x - y)/x;    % definice ODE
cond = y(1) == 3;                  % pocatecni podminka
reseni_sym = dsolve(ode, cond);    % analyticke reseni

% Prevod na funkci pro dosazovani hodnot
y_fun = matlabFunction(reseni_sym);

% --- Parametry ulohy ---
x0 = 1;
y0 = 3;
xN = 3;

% --- Numericke reseni ODE45 ---
prava_strana = @(x,y) (4*x - y)/x;
[t_ode, reseni_ode] = ode45(prava_strana, [x0, xN], y0);
```

```
21
22 % --- Eulerova metoda ---
23 h = 1;                                          % krok
24 x_ove = x0:h:xN;                                % body na ose x
25 y_ove = zeros(1, length(x_ove));                % vektor pro reseni
26 y_ove(1) = y0;                                  % pocatecni podminka
27 for k = 2:length(x_ove)
28     y_ove(k) = y_ove(k-1) + h*prava_strana(x_ove(k-1), y_ove(k-1)
       );
29 end
30
31 % --- Vyhodnoceni presneho reseni ---
32 y_sym = y_fun(x_ove);
33
34 % --- Graficke porovnani ---
35 figure;
36 hold on
37 grid on
38 plot(x_ove, y_sym, 'r-o')                 % symbolicke reseni
39 plot(t_ode, reseni_ode, 'b--')            % reseni ode45
40 plot(x_ove, y_ove, 'g-s')                 % Eulerova metoda
41 title('Reseni Cauchyovy ulohy')
42 legend('symbolicke','ode45','Euler')
43 hold off
```

Kapitola 10

Materiálové vědy

10.1 Mřížky a krystalografie

1. Vytvoření jednoduché 2D mřížky

```
[X,Y] = meshgrid(0:0.1:1,0:0.1:1);
Z = sin(pi*X).*sin(pi*Y);
surf(X,Y,Z)
xlabel('X'); ylabel('Y'); zlabel('Z')
```

Generování dvourozměrné mřížky s periodicitou, vhodné pro vizualizaci potenciálních energií.

2. Generování 3D kubické mřížky

```
[x,y,z] = meshgrid(0:1:4,0:1:4,0:1:4);
scatter3(x(:),y(:),z(:),'filled')
axis equal
```

Trojrozměrná kubická mřížka, základní model pro kovové krystaly.

3. Kubická supermřížka s více typy atomů

```
[x,y,z] = meshgrid(0:1:3,0:1:3,0:1:3);
types = mod(x+y+z,2);
scatter3(x(:),y(:),z(:),100,types(:),'filled')
axis equal
```

Simulace slitin s více typy atomů.

4. Hexagonální mřížka 2D

```
[X,Y] = meshgrid(0:1:5,0:sqrt(3)/2:5*sqrt(3)/2);
X(2:2:end,:) = X(2:2:end,:) + 0.5;
scatter(X(:),Y(:))
axis equal
```

Hexagonální mřížka pro vrstvy grafitu.

5. 2D deformace mřížky

```
[X,Y] = meshgrid(0:0.1:1,0:0.1:1);
Z = 0.1*X.^2 - 0.05*Y.^2;
surf(X,Y,Z)
```

Vizualizace lokální deformace mřížky.

6. 3D mřížka s barevným rozlišením vzdálenosti

```
[x,y,z] = meshgrid(0:1:4,0:1:4,0:1:4);
dist = sqrt((x-2).^2 + (y-2).^2 + (z-2).^2);
scatter3(x(:),y(:),z(:),[],dist(:),'filled')
axis equal
colorbar
```

Barevné odlišení vzdálenosti od středu mřížky.

10.2 Poruchy a defekty

7. Simulace vakancí v mřížce

```
N = 100;
vacancy = randperm(N,5);
atoms = ones(1,N);
atoms(vacancy) = 0;
stem(1:N,atoms)
```

Náhodné vakance v mřížce.

8. Dislokace 2D

```
[X,Y] = meshgrid(1:10,1:10);
U = zeros(size(X));
U(5,5) = 1;
surf(X,Y,U)
```

Vizualizace bodové dislokace.

9. Hrana dislokace

```
[X,Y] = meshgrid(0:1:10,0:1:10);
U = 0.05*sin(pi*X/10).*cos(pi*Y/10);
surf(X,Y,U)
```

Simulace hrany dislokace.

10. Náhodné poruchy 1D mřížky

```
N = 50; defects = zeros(1,N);
defects(randperm(N,5)) = 1;
bar(defects)
```

Náhodné rozmístění defektů.

11. Pravděpodobnost vakancí

```
N = 1000; p = 0.01;
vacancies = rand(1,N)<p;
histogram(vacancies,2)
```

Statistický model pravděpodobnosti vzniku vakancí.

12. Monte Carlo simulace dislokace

```
N = 20; lattice = zeros(N);
for k=1:50
    i = randi(N); j = randi(N);
    lattice(i,j) = 1;
end
imagesc(lattice)
colormap(gray)
```

Monte Carlo simulace náhodných dislokací.

10.3 Fázové diagramy

13. Diagram Fe–Fe_3C

```
T = 700:50:1500;
C = linspace(0,6.7,100);
phase = zeros(length(C),length(T));
for i=1:length(T)
    for j=1:length(C)
        if C(j)<0.8
            phase(j,i)=1;
        elseif C(j)<2.1
            phase(j,i)=2;
        else
            phase(j,i)=3;
        end
    end
end
imagesc(T,C,phase); colorbar
xlabel('T [K]'); ylabel('C [%]')
```

Rozlišení fází v Fe–Fe3C diagramu.

14. Eutektický bod

```
C = linspace(0,6.7,100);
T_eut = 1147;
C_eut = 4.3;
plot(C,T_eut*ones(size(C)),'r--')
hold on
```

```
plot(C_eut*ones(size(C)),linspace(700,1500,100),'b--')
```

Vizualizace eutektického bodu.

15. Napětí–deformace lineární

```
stress = 0:10:500;
strain = 0.002*stress;
plot(stress,strain)
```

Lineární model Hookeova zákona.

16. Napětí–deformace nelineární

```
stress = 0:10:500;
strain = 0.002*stress + 1e-5*stress.^2;
plot(stress,strain)
```

Simulace nelineárního chování materiálu.

17. Gibbsova energie

```
T = 300:10:1500;
G = -80000 + 50*T;
plot(T,G)
```

Závislost Gibbsovy energie na teplotě.

18. Difuze 1D

```
D = 1e-9; L=1e-3; dx=1e-5; dt=1;
x = 0:dx:L; C = zeros(size(x)); C(round(end/2))=1;
for t=1:500
    C(2:end-1) = C(2:end-1) + D*dt/dx^2*(C(3:end)-2*C(2:end-1)+C
       (1:end-2));
end
plot(x,C)
```

Simulace difuze v 1D.

10.4 Monte Carlo a statistika

19. FFT signálu

```
signal = sin(2*pi*10*(0:0.001:1)) + 0.5*randn(1,1001);
Y = fft(signal);
f = (0:length(Y)-1)/(length(Y)*0.001);
plot(f,abs(Y))
```

Analýza frekvenčního spektra.

20. Histogram vibrací

```
data = randn(1,1000);
histogram(data,30)
```

Statistické rozdělení náhodných vibrací.

21. Monte Carlo mřížka

```
N = 20; lattice = zeros(N);
for k=1:500
    i = randi(N); j = randi(N);
    lattice(i,j) = mod(lattice(i,j)+1,2);
end
imagesc(lattice)
colormap(gray)
```

Monte Carlo simulace 2D mřížky.

22. Náhodné rozdělení atomů

```
atoms = rand(1,1000)<0.3;
histogram(atoms)
```

Náhodné rozdělení atomů v materiálu.

23. Statistický rozptyl

```
x = randn(1,1000);
y = randn(1,1000);
scatter(x,y)
```

Vizualizace rozptylu vlastností.

24. Korelace vlastností

```
x = randn(1,1000); y = 0.5*x + randn(1,1000)*0.2;
plot(x,y,'.')
```

Korelace mezi dvěma vlastnostmi.

25. Monte Carlo difuze

```
% Monte Carlo simulace difuzního pohybu částic (1D)
N = 100;         % počet částic
steps = 500;     % počet kroků
pos = zeros(1,N);  % počáteční poloha částic

% Simulace náhodného pohybu
for t = 1:steps
```

```
    pos = pos + randi([-1,1],1,N);  % krok vlevo, vpravo nebo zů
        stat
end

% Histogram konečných poloh
h = histogram(pos, 'Normalization','pdf', ...
              'FaceColor',[0.2 0.6 0.8], 'EdgeColor','k');
hold on;

% --- Teoretická normální distribuce ---
mu = 0;
sigma = sqrt((2/3)*steps);

x_vals = linspace(min(pos)-10, max(pos)+10, 200);
pdf_vals = normpdf(x_vals, mu, sigma);

plot(x_vals, pdf_vals, 'r-', 'LineWidth',2);

% Popisky a vzhled
xlabel('Konečná poloha částice (x)', 'FontSize',12);
ylabel('Pravděpodobnostní hustota', 'FontSize',12);
title('Difuzní pohyb částic simulovaný metodou Monte Carlo', '
   FontSize',14);

legend('Simulace (histogram)', 'Teoretické N(0,\sigma^2)', ...
       'Location','best');

grid on;
set(gca, 'FontSize',12, 'LineWidth',1.2, 'Box','on');
```

Difuzní pohyb částic simulovaný Monte Carlo.

26. Statistické napětí

```
stress = randn(1,1000)*50 + 200;
histogram(stress,30)
```

Statistické rozdělení napětí.

27. Analýza vlastních frekvencí

```
A = randn(5);
eig(A)
```

Výpočet vlastních frekvencí matice.

28. Náhodné fázové pole

```
N=50; phase = rand(N);
imagesc(phase)
colormap(jet)
```

Simulace náhodného fázového pole.

29. Difuze dvou složek

```
N=50; A=zeros(N); B=zeros(N); A(round(N/2))=1; B(round(N/2))=1;
for t=1:100
    A(2:end-1) = A(2:end-1)+0.01*(A(3:end)-2*A(2:end-1)+A(1:end
       -2));
    B(2:end-1) = B(2:end-1)+0.01*(B(3:end)-2*B(2:end-1)+B(1:end
       -2));
end
plot(A); hold on; plot(B)
```

Difuze dvou komponent v 1D systému.

30. Histogram fází

```
phase = rand(1,1000);
histogram(phase,20)
```

Statistické rozdělení fází.

10.5 Pokročilá vizualizace

31. Povrchová vizualizace

```
[X,Y] = meshgrid(-2:0.1:2,-2:0.1:2);
Z = exp(-X.^2-Y.^2).*sin(3*X).*cos(3*Y);
surf(X,Y,Z)
```

Vizualizace komplexního povrchu.

32. 3D histogram

```
data = randn(1000,3);
hist3(data(:,1:2),[20,20])
```

3D histogram rozložení částic.

33. Vektorové pole

```
[x,y] = meshgrid(-2:0.2:2,-2:0.2:2);
u = -y; v = x;
quiver(x,y,u,v)
```

Vizualizace vektorového pole.

34. Gradient fázového pole

```
[X,Y] = meshgrid(0:0.1:2*pi,0:0.1:2*pi);
Z = sin(X).*cos(Y);
[FX,FY] = gradient(Z,0.1,0.1);
quiver(X,Y,FX,FY)
```

Vektorový gradient fázového pole.

35. Povrchová topologie

```
[X,Y] = meshgrid(-3:0.1:3,-3:0.1:3);
Z = X.*exp(-X.^2-Y.^2);
surf(X,Y,Z)
```

Vizualizace povrchových vlastností materiálu.

36. 3D scatter s barvou

```
[x,y,z] = rand(3,100);
c = x+y+z;
scatter3(x,y,z,50,c,'filled')
```

Barevně odlišené body v 3D prostoru.

37. Povrchová deformace

```
[X,Y] = meshgrid(0:0.1:2,0:0.1:2);
Z = sin(pi*X).*cos(pi*Y) + 0.1*rand(size(X));
surf(X,Y,Z)
```

Simulace lokálních povrchových deformací.

38. Mřížka s odchylkami

```
[x,y] = meshgrid(0:0.1:1,0:0.1:1);
x = x + 0.02*rand(size(x));
y = y + 0.02*rand(size(y));
scatter(x(:),y(:))
```

Vizualizace nepravidelností mřížky.

39. 2D fázová mapa

```
phase = rand(50);
imagesc(phase)
colormap(jet)
colorbar
```

Fázová mapa, simulace lokálních vlastností.

40. Kombinace mřížky a deformace

```
[X,Y] = meshgrid(0:0.1:1,0:0.1:1);
Z = sin(pi*X).*sin(pi*Y) + 0.05*rand(size(X));
surf(X,Y,Z)
```

Kombinace periodické mřížky a náhodné deformace.

10.6 Identifikace slitiny podle fyzikálních a mechanických vlastností

Příprava dat

```

Nazvy = { 'Ocel uhlíková','Nerezová ocel','Hliníková slitina','Bronz', ...
          'Titanová slitina','Měď','Mosaz','Ocel nástrojová','Ocel legovaná', ...
          'Aluminium 7075','Titan Grade 5','Nikl','Zinek','Ocel P235', ...
          'Ocel S355','Ocel 316L','Hliník 6061','Hliník 2024','Hliník 5083', ...
          'Hliník 5052','Hliník 7075','Hliník 1100','Hliník 3003', ...
          'Hliník 2024-T3','Titan Beta','Titan Alpha','Bronz CuSn10', ...
          'Mosaz CuZn30','Ocel C45','Ocel 42CrMo'};

Hustota   = [7850,8000,2700,8800,4500,8960,8530,7850,7850,2810, ...
             4430,8900,7140,7850,7850,8000,2700,2780,2650,2650, ...
             2810,2700,2730,2780,4430,4500,8800,8530,7850,7800];

ModulPruz = [210,200,70,100,115,110,100,210,215,71, ...
             114,200,100,205,210,200,68,73,70,72, ...
             71,69,70,73,112,110,95,100,208,210];

MezPevn   = [400,520,310,450,900,210,310,600,580,572, ...
             950,360,140,430,470,520,310,325,300,305, ...
             580,290,305,330,940,910,450,320,470,500];

MezKluz   = [250,300,240,200,880,70,220,580,540,503, ...
             880,310,130,400,450,310,280,290,260,265, ...
             503,270,275,295,860,850,190,200,450,470];

TepVodiv  = [50,16,180,50,7,400,110,48,45,130, ...
             6.7,90,116,50,48,16,167,175,135,135, ...
             130,217,130,175,6.8,7.1,50,110,50,48];

SpecC     = [460,500,900,380,520,385,380,460,470,875, ...
             522,440,390,460,465,500,900,910,890,885, ...
             875,897,890,905,518,515,380,382,460,465];
```

Vytvoření tabulky (řádky = jednotlivé slitiny)

```
tab = table(Nazvy', Hustota', ModulPruz', MezPevn', MezKluz', TepVodiv', SpecC', ...
```

```
    'VariableNames', {'Nazev','Hustota','ModulPruz','MezPevn','
      MezKluz','TepVodiv','SpecC'});
```

Vstup uživatele (neznámé hodnoty = NaN)

```
vstupy.Hustota   = 7800;
vstupy.ModulPruz = 210;
vstupy.MezPevn   = NaN;   % neznám     ignoruje se
vstupy.MezKluz   = 400;
vstupy.TepVodiv  = NaN;
vstupy.SpecC     = 500;
```

Hledání shod

```
[shody, distances] = najdiSlitinu(tab, vstupy, 'MaxMatches', 3);

disp('Nejlepší shody:');
disp(shody);
disp('Vzdálenosti (normalizované min-max):');
disp(array2table(distances, 'VariableNames',{'Distance'}));
```

Radar graf (normalizované osy)

```
plotRadar(shody, vstupy, tab);
```

Funkce: najdiSlitinu

```
function [shody, distances] = najdiSlitinu(tab, vstupy, varargin)
 % [shody, distances] = najdiSlitinu(tab, vstupy, 'MaxMatches', k
    )
  % Robustní vyhledání nejbližších slitin podle zadaných číselný
     ch vlastností.
  %-tab: table s proměnnými (sloupci) odpovídajícími jménům v '
     vstupy'
  %-vstupy: struct, pole mohou být NaN (znamená "ignoruj")
  %-vrací tabulku 'shody' a vektor vzdáleností (po normalizaci
     min-max)

  %---parametry
  p = inputParser;
  addParameter(p, 'MaxMatches', 3, @(x) isnumeric(x) && isscalar(
     x) && (x>=1));
  parse(p, varargin{:});
  maxMatches = double(p.Results.MaxMatches);

  %---vyber použitelných polí (ne-NaN a existují v tabulce)
```

```
15    allFld = fieldnames(vstupy);
16    isNotNaN = ~structfun(@(x) any(isnan(x)), vstupy); % true pro
         pole, která nejsou NaN
17    existInTab = ismember(allFld, tab.Properties.VariableNames);
18    fldNames = allFld(isNotNaN & existInTab);
19    if isempty(fldNames)
20        shody = tab([],:); distances = [];
21        warning('Žádná použitelná pole k porovnání (vstupy all NaN
             nebo chybí v tab).');
22      return;
23    end
24   %---extrakce numerických dat z tabulky pro vybraná pole
25    Mraw = tab{:, fldNames}; % n x k
26    if ~isnumeric(Mraw)
27        error('Vybrané sloupce tabulky musí být numerické.');
28    end
29   %---odstranění řádků s NaN v těchto sloupcích
30    validRows = all(~isnan(Mraw), 2);
31    if ~all(validRows)
32         warning('Některé řádky tab obsahují NaN v požadovaných
              sloupcích budou vynechány.');
33        Mraw = Mraw(validRows, :);
34    end
35   origIdx = find(validRows); % indexy řádků v původní tabulce
36  %---min-max normalizace sloupců (škálováno do [0,1])
37    mn = min(Mraw, [], 1);
38    mx = max(Mraw, [], 1);
39    den = mx-mn;
40    den(den == 0) = 1; % pokud konst. sloupec, zabráníme dělení
         nulou
41    M = (Mraw-mn) ./ den;
42   %---vstupní vektor a jeho normalizace
43    v = zeros(1, numel(fldNames));
44    for i = 1:numel(fldNames)
45        v(i) = vstupy.(fldNames{i});
46    end
47    v = (v-mn) ./ den;
48   %---spočti eukleidovskou vzdálenost (po normalizaci)
49    diffs = M-v; % n x k
50    distancesAll = vecnorm(diffs, 2, 2); % n x 1
51   %---seřaď a vyber top-k
52    [sortedD, idxLocal] = sort(distancesAll, 'ascend');
53    k = min(maxMatches, numel(sortedD));
54    selIdxLocal = idxLocal(1:k);
55    selOrigRows = origIdx(selIdxLocal); % map back to original
         table rows
56   shody = tab(selOrigRows, :);
57   distances = sortedD(1:k);
58
59  end
```

Funkce: plotRadar

```
function plotRadar(shody, vstupy, tab)
% Vykreslí radar (spider) graf pro vstup a nalezené shody.
% Osy jsou normalizované podle min-max z celé tabulky
% pro korektní srovnání.

% vyber stejná pole jako ve funkci najdiSlitinu
allFld = fieldnames(vstupy);
isNotNaN = ~structfun(@(x) any(isnan(x)), vstupy);
fldNames = allFld(isNotNaN & ismember(allFld, tab.Properties.
    VariableNames));
kFields = numel(fldNames);
if kFields == 0
    warning('plotRadar: žádná pole k vykreslení.');
    return;
end

% referenční rozsahy z celé tabulky (nejen z shod) pro
    konzistentní škálování
Mfull = tab{:, fldNames};
mn = min(Mfull, [], 1);
mx = max(Mfull, [], 1);
den = mx - mn;
den(den == 0) = 1;

% vstup
v = zeros(1, kFields);
for i = 1:kFields
    v(i) = vstupy.(fldNames{i});
end
vnorm = (v - mn) ./ den;

% shody
nShody = height(shody);
data = zeros(nShody+1, kFields);
data(1,:) = vnorm;
for j = 1:nShody
    row = shody{j, fldNames};
    data(j+1,:) = (row - mn) ./ den;
end

% uzavření polygonu
data = [data data(:,1)];

% úhly
theta = (0:kFields) * 2*pi / kFields;

% grafika (polaraxes místo obyč. axes)
figure;
pax = polaraxes;
```

```
hold(pax, 'on');
colors = lines(nShody+1);
markers = {'o','s','^','d','x'};
names = cell(nShody+1,1);
names{1} = 'Vstup';
for j = 1:nShody
    names{j+1} = shody.Nazev{j};
end

% vykreslení polygonů
for r = 1:nShody+1
    h = polarplot(pax, theta, data(r,:), 'LineWidth', 2, 'Color',
        colors(r,:));
    h.Marker = markers{min(r,numel(markers))};
    h.DisplayName = names{r};
end

% popisky jednotlivých os (fldNames)
rt = 1.1; % vzdálenost textů od jednotkového kruhu
for i = 1:kFields
    ang = theta(i);
    x = rt * cos(ang);
    y = rt * sin(ang);
    text(pax, x, y, fldNames{i}, 'HorizontalAlignment','center',
       ...
        'VerticalAlignment','middle', 'FontSize', 10, 'FontWeight
           ','bold');
end

title(pax, 'Porovnání vstupu a nalezených slitin (normalizováno)'
   );
legend(pax, 'Location','bestoutside');
hold(pax, 'off');

end
```

Shrnutí významu radarových grafů

Radarové grafy představují vizuální srovnání mezi hledaným vzorkem a nejbližšími nalezenými slitinami z databáze. Každá osa grafu odpovídá jedné fyzikální nebo mechanické vlastnosti, která byla při identifikaci vzorku zadána. Hodnoty vstupního vzorku jsou zobrazeny jako uzavřený polygon (většinou označený kroužky), zatímco jednotlivé kandidátní slitiny jsou znázorněny jinými tvary a barvami.

- **Graf s nejlepší shodou** : porovnává pouze vstupní vzorek a slitinu, která vykázala nejmenší Eukleidovskou vzdálenost (tj. nejvyšší podobnost). Pokud se tvary polygonů výrazně překrývají, lze konstatovat vysokou shodu zadaných parametrů.
- **Graf s více kandidáty** : tento graf zobrazuje vstupní vzorek a až tři nejlepší shody z databáze. Umožňuje přímé vizuální porovnání toho, jak moc se jednotlivé kandidátní slitiny liší od vzorku a od sebe navzájem.

Radarová vizualizace tak poskytuje intuitivní přehled o míře podobnosti mezi hledaným materiálem a databází známých slitin. Zatímco samotný algoritmus pracuje s číselnou vzdáleností, grafická forma ukazuje, ve kterých konkrétních vlastnostech je shoda nejvyšší a kde naopak vznikají odchylky.

Kapitola 11

Poznámky

11.0.1 Terminologie funkcí v Matlabu

V následující tabulce jsou shrnuty základní termíny související s funkcemi v prostředí Matlab.

Tabulka 11.1: Přehled terminologie pro funkce v Matlabu

Anglický termín	**Doslovný překlad**	**Doporučený český ekvivalent**	**Poznámka**
function	funkce	funkce	základní pojem
script	skript	skript	soubor bez definice funkcí, běží v *base workspace*
anonymous function	anonymní funkce	anonymní funkce	jednorázová funkce, např. `@(x) x^2`
nested function	vnořená funkce	vnořená funkce	funkce definovaná uvnitř jiné funkce
inner function	vnitřní funkce	vnitřní funkce	synonymum k „nested function“
outer function	vnější funkce	vnější funkce	funkce, která obsahuje vnořenou funkci
parent function	rodičovská funkce	nadřazená funkce	formálnější výraz, méně užívaný
local function	lokální funkce	lokální funkce	definovaná na konci souboru, přístupná jen v něm
subfunction	podfunkce	podfunkce	starší termín, dnes spíše „lokální funkce“
workspace	pracovní prostor	pracovní prostor	sada proměnných a jejich hodnot