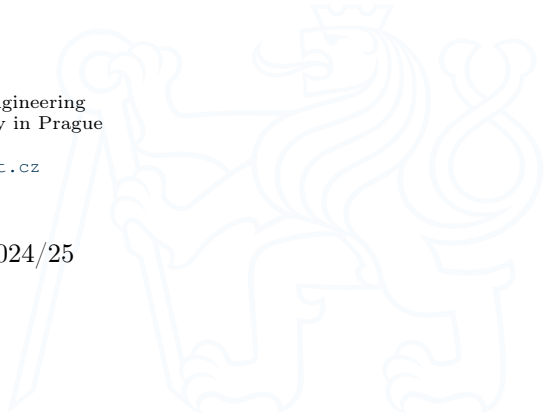
Lecture 12: Lists, Rules and Patterns.

A8B17CAS

Jozef Lukáč

Department of Radio Engineering
Czech Technical University in Prague
Czech Republic
lukacjo1@fel.cvut.cz

December 2
Winter semester 2024/25



Outline

1. Creation of matrices and vectors.
2. Accessing parts of matrices/vectors/expressions.
3. Rearranging lists.
4. Max, MaximalBy, DeleteDuplicates.
5. Lists as Sets.
6. Functions for testing properties of numbers.
7. Patterns
8. Functions for testing structural properties of expressions.
9. Rules
   9.1 Rule vs. RuleDelayed, Set vs. SetDelayed
10. Putting constraints on patterns and transformation rules.
11. Cases, Count, Position, Select

Creation of matrices and vectors.

Common commands to create a matrix and a vector:

- **Range**, **Table**, (**Array**), **IdentityMatrix**, **DiagonalMatrix**, **Subdivide**, *e.g.*

  {Range[5], Range[-3,1], Range[2,13,3]} →

  ```
  {{1, 2, 3, 4, 5}, {-3, -2, -1, 0, 1}, {2, 5, 8, 11}}
  ```

  in MATLAB it would be `1:5, -3:1, 2:3:13`.
- Use function **f[n]**, **f[m,n]**, **f[m,n,o]**, ... to specify each element of a vector/matrix/multidimensional array, *e.g.*:

  Table[f[n],{n,5}] → `{f[1], f[2], f[3], f[4], f[5]}`

  Table[i^2 Sqrt[j],{i,2,4},{j,3, 8, 2}] →

  $\{\{4\sqrt{3}, 4\sqrt{5}, 4\sqrt{7}\}, \{9\sqrt{3}, 9\sqrt{5}, 9\sqrt{7}\}, \{16\sqrt{3}, 16\sqrt{5}, 16\sqrt{7}\}\}$
- **Table[0,{n1Max},n2Max,...]** is equivalent to `zeros(n1Max,n2Max,...)` in MATLAB.
- **Table[1,{n1,n1Max},{n2Max},...]** is equivalent to `ones(n1Max,n2Max,...)` in MATLAB.

Creation of matrices and vectors.

- `Table[RandomReal[],n1Max,{n2Max},...]`, resp. `RandomReal[{0,1},{n1Max,n2Max,...}]` is equivalent to `rand(n1Max,n2Max,...)` in MATLAB.
- For the generation of a random number from a general distribution, use `RandomVariate`, *e.g.*
  `RandomVariate[NormalDistribution[],{n1Max,n2Max,...}]` is equivalent to `randn(n1Max,n2Max,...)` in MATLAB.
- `IdentityMatrix[n]` is equivalent to `eye(n)` in MATLAB.
- To create a diagonal matrix from a vector, use `DiagonalMatrix[vec,n]` (equivalent to `diag(vec,n)` in MATLAB).
- To extract a diagonal vector from a matrix, use `Diagonal[mat,n]` (equivalent to `diag(mat,n)` in MATLAB).
- Equivalent to MATLAB `linspace(a,b,n)` is `Subdivide[a,b,n-1]`.
- To concatenate matrices/expressions, use `Join`, *e.g.*

```
mat1 = {{1, 2, 3}, {4, 5, 6}}; mat2 = {{7, 8, 9}, {10, 11, 12}};
MatrixForm /@ {mat1,mat2,mat1~Join~mat2,Join[mat1,mat2,1], Join[mat1,mat2,2]}
```

- Get dimensions of matrix/expression by `Dimensions`, *e.g.* `Dimensions[mat1]`.

Create matrix, example

- Create the following matrix

$$\begin{bmatrix} -2 & -1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 5 & 0 & 0 & r_1 & r_2 \\ 0 & 6 & 0 & r_3 & r_4 \\ 0 & 0 & 7 & r_5 & r_6 \end{bmatrix}.$$

where the r_i are random number from the uniform distribution on $(0, 1)$. You can define auxliary join command: `myJoin = Join[#1, #2, 2] &;` (for horizontal concatenation)

```
myJoin = Join[#1, #2, 2] &;
...
% // MatrixForm
```

Exercise

Accessing parts of matrices/vectors/expressions.

We can access different parts of matrices/expressions by the following commands:

- **Part[expr,idc]**, resp. **expr[[idc]]** – by specifying the index of a part, *e.g.*
  Part[mat1,2, {2, 3}], → `(3 x + Sin[x^3])[[2, 1, {1, 2}]]`
- **Take[expr,n]** – Get the first `n` elements from the expression/list. *e.g.*
  Take[a^8 b c^4 d,2] → `a^8 b`
- **Drop[list,n]** – Returns list with its first `n` elements dropped, *e.g.*
  Drop[{4, 5, 9, -2}, 2] → `{9, -2}`
- **Most[expr]** – Returns expression with its last element removed, *e.g.*
  Most[9 + x^2 + Sin[x]] → `9+x^2`
- **Last[expr]** – Returns the last part of expression, *e.g.*
  Last[f[a^2, b, c^4, d, e]] → `e`
- **First[expr]** – Gives the first part of expression, *e.g.*
  First[a^8 b c] → `a^8`
- **Rest[expr]** – Returns the expression with the first element removed, *e.g.*
  Rest[a^8 b c] → `b c`

Accessing parts of matrices/vectors/expressions – examples

Exercise

- Let's have a list of polynomials. Transform each polynomial by omitting first and last summand. Implement 2 ways – one using indexing, and one without indexing. *E.g.* `x + 2 y + 3 z` should transform to `2 y`.

```
nMax = 5;
Table[(x + 2 y + 3 z)^n // Expand, {n, nMax}];
% // Column
...
...
```

- Rewrite the following code without using indexing.

```
nMax = 20;
IntegerDigits /@ Round@(Pi^Range[7, nMax]);
% // Column
#[[-3 ;; -1]] & /@ %% // Column
#[[-1]] & /@ %%%
#[[4 ;;]] & /@ %%%% // Column
```

Rearranging lists.

Some useful commands to rearrange lists/expressions.

- `Sort[list,orderingFuncion]` – sorts list/expression according to the ordering function (takes 2 elements and returns True/False).
- `SortBy[list,f]` – sorts list in the order defined by applying `f` to each element.
- `RotateLeft[expr,n]` – cycles the elements in expr `n` positions to the left.
- `RotateRight[expr,n]` – cycles the elements in expr `n` positions to the right.
- `Transpose[list]` – transposes the first two levels in list.
- Examples,

```
Sort[f[1,3,2]]                                  f[1,2,3]
SortBy[{{a,1},{{3,1},3},{x^2,2}},Last]          {{a,1},{x^2,2},{{3,1},3}}
RotateRight[{5,-1,4,2},2]                       {4,2,5,-1}
```

Max, MaximalBy, DeleteDuplicates.

- **`Max[list]`** – returns maximal number from the list of numbers.
- **`MaximalBy[list,f]`** – returns a list of elements for which `f[e_i]` is maximal.
- **`DeleteDuplicates[list]`** – deletes all duplicates from list. **`Sort@DeleteDuplicates[list]`** and **`Union[Sequence @@ {#} & /@ list]`** are equivalent to `unique(list)` in MATLAB.
- Examples,

```
Max[{5,4,6,-2,3}]                                  6
MaximalBy[{{a,1},{{3,1},3},{x^2,2}},Last]          {{3,1},3}
DeleteDuplicates[{2,3,-1,2,5,6,3,2,8}]             {2,3,-1,5,6,8}
```

Max, Sort – examples.

- Assume you have a matrix of numbers. 1) sort its rows according to the third column (use **Sort**); 2) sort its rows according to the last column (use **SortBy**); 3) return a row that has the highest number at second position (use **MaximalBy**).

```
SeedRandom[2024]
nMax = 6;
matA = RandomInteger[{-10, 10}, {nMax, 4}]
% // MatrixForm
... (*Sort*)
... (*SortBy*)
... (*MaximalBy*)
MatrixForm /@ {%%%%, %%%, %%, %} (*show outputs*)
```

$$\begin{pmatrix} 5 & 5 & -5 & -7 \\ 0 & 7 & 4 & -4 \\ 4 & 1 & 8 & -9 \\ 0 & 5 & 9 & 1 \\ 0 & -10 & -2 & 3 \\ -1 & -3 & 2 & 0 \end{pmatrix}, \begin{pmatrix} 5 & 5 & -5 & -7 \\ 0 & -10 & -2 & 3 \\ -1 & -3 & 2 & 0 \\ 0 & 7 & 4 & -4 \\ 4 & 1 & 8 & -9 \\ 0 & 5 & 9 & 1 \end{pmatrix}, \begin{pmatrix} 4 & 1 & 8 & -9 \\ 5 & 5 & -5 & -7 \\ 0 & 7 & 4 & -4 \\ -1 & -3 & 2 & 0 \\ 0 & 5 & 9 & 1 \\ 0 & -10 & -2 & 3 \end{pmatrix}, (0\ 7\ 4\ -4)$$

Exercise

Lists as Sets.

To use lists as sets, the following commands are useful.

- `Union[list1,list2,...]` – gets a union of the lists.
- `Intersection[list1,list2,...]` – gets the intersection of all the lists.
- `Complement[allElems,list1,list2,...]` – gives element that are not present in any list.
- Examples

```
Union[f[6,3,4],f[1,3]]                          f[1,3,4,6]
Intersection[{5, 3, 6, 1, 3}, {3, 6, 4}]        {3,6}
Complement[Range[6], {5, 3}, {2, 3}]            {1,4,6}
```

Functions for testing properties of numbers.

Test functions in MATHEMATICA usually end with `Q` (query, question) command/function.
And return **`True`** or **`False`**.

- **`IntegerQ`** – whether the number is integer.
- **`EvenQ`** – whether the number is even.
- **`PrimeQ`** – whether the number is prime.
- **`VectorQ`** – whether the input is a simple list (without nested lists).
- **`MatrixQ`** – whether the input is a matrix – list of lists (of same length).
- **`NumericQ`** – whether the input object is numeric.
- Examples:

```
IntegerQ /@ {1,2/3,Sqrt[3],-4}                    {True, False, False, True}
MatrixQ /@ {5, {1,2},x^2, {{1,2},{3,4}}}          {False, False, False, True}
```

Patterns

FUNDAMENTAL PRINCIPLE of MATHEMATICA: *Take any expression, and apply transformation rules until the result no longer changes.*

- *Pattern is representation of a group/class of expressions*, *e.g.* `f[_]` means `f[anyExpression]`. *E.g.* `Cases[5+x+f[x]+f[y^2], f[_]]` → `{f[x],f[y^2]}`
- Similar to *regular expressions*[1] for string matching.
- Examples of patterns:

`_`	any single expression,
`x_` or `x:_`	any single expression to be named `x`,
`__`	any *sequence* of one or more expressions,
`x__h` or `x:__h`	sequence of expressions, all of whose heads are `h`,
`___`	any sequence of zero or more expressions,
`x___` or `x:___`	any sequence of zero or more expressions named `x`,
`x___h` or `x:___h`	sequence of zero or more expressions, all of whose heads are `h`,
`f[n_]` or `f[n:_]`	`f` with any argument, named `n`,
`f[n_,m_]` or `f[n:_,m:_]`	`f` with two arguments, named `n`and `m`,
`x^n_` or `x^n:_`	`x` to any power, with the power named `n`,

[1] https://en.wikipedia.org/wiki/Regular_expression

Patterns (2)

`x_^n_` or `x:_^n_`	any expression to any power,
`a_ + b_` or `a_ + b:_`	a sum of two expressions,
`{_,a2_}` or `{_,a2:_}`	a list of two expressions, second named `a2`
patt$_1$`|`*patt*$_2$`|...`	pattern alternatives,
	e.g. `{a,b,c,d} /.(a|b)->p` → `{p,p,c,d}`
patt`..` or `Repeated[`*patt*`]`	repeated pattern, *e.g.* `Cases[{f[a],f[a,b,a],f[a,a,a]},f[a..]]` → `{f[a],f[a,a,a]}`

- patterns for objects with specified Heads:

`x_h`	an expressions with the head `h`,
`x_Integer`	an expressions with head `Integer`,
`x_Real`	an expressions with head real number,
`x_Complex`	a complex number,
`x_List`	a list,
`x_Symbol`	a symbol

- *To name a whole pattern*, we use a colon, `:`, *e.g.* `x:_Integer`, `a:f[n_]`, `p:x_^n_`.

Patterns (3)

- Patterns can be used in commands and are often used in rules, *e.g.* **Cases** chooses subexpressions that match the pattern (and can possibly apply a rule on it)[2].

```
v1={4,{1.3,1},{3,1,-1},{2,1},-3,{1.,-0.1}};
Cases[%, _Integer]                                  {4,-3}
Cases[%%, _List]                                    {{1.3,1},{3,1,-1},{2, 1},{1.,-0.1}}
Cases[%%%, {__Integer}]                             {{3,1,-1},{2, 1}}
Cases[v1, list:{_, _}:>list^2]                      {{1.69, 1}, {4, 1}, {1., 0.01}}
v1 /. {a_, _}:>a                                    {4, 1.3, {3, 1, -1}, 2, -3, 1.}
Replace[v1, l_List:>First[l], {1}]                  {4, 1.3, 3, 2, -3, 1.}
g[b_Integer]:= b^2; g[b_Real]:=Round[b]; g /@ %      {16, 1, 9, 4, 9, 1}
```

[2] more info on patterns at https://reference.wolfram.com/language/tutorial/Patterns.html

Patterns, example

- Use the list `v1` (from the previous slide). Use `Cases` with a level specification to get integers 1) at the first level 2) at the second level.

```
v1={4, {1.3, 1}, {3, 1, -1}, {2, 1}, -3,{1.,-0.1}}
```

```
... → {4,-3}
```

```
... → {1, 3, 1, -1, 2, 1}
```

- Use the list `v1` and the command `Cases` to choose only 2-element lists in `v1` such that first is a real number.

```
... → {{1.3, 1}, {1., -0.1}}
```

Exercise

Sequence and Nothing.

- **`Sequence[...]`** represents a *headless* sequence of arguments

  `{a, b, Sequence[c, d, e], f, g}` → `{a, b, c, d, e, f, g}`

  `g[{3, -1}] /. List -> Sequence` → `g[3, -1]`

- **`Nothing`** can serve for deleting elements, *e.g.*

  `{1, 2, 3, 6, 3, 4, 6, 4, 2, 3} /. {3 -> Nothing}` → `{1, 2, 6, 4, 6, 4, 2}`

Functions for testing structural properties of expressions.

- **Equal**, resp. == – true if both sides are identical.
- **OrderedQ[list]** – checks whether the list is sorted.
- **MemberQ[list,form]** – checks whether an element from a list matches the form.
- **FreeQ[expr,form]** – checks whether no subexpression matches the form.
- **MatchQ[expr,form]** – true if the pattern form matches expr.
- **ValueQ[expr]** – whether a value has been defined for expr.
- **AtomQ[expr]** – true if the expr is an atomic expression (cannot be divided into subexpression).
- Examples:

```
OrderedQ[{1,x^3,x^4}]                          True
expr = Sin[x^3] + x y; MemberQ[expr, y, {2}]   True
{FreeQ[expr, _^3], FreeQ[expr, _^4]}           {False, True}
```

Rules

- Rules represent a substitution. The first part can be a general pattern. The second part usually does not contain a pattern. *E.g.* `rule1=x->x^2` (`Rule`), `rule2=x_:>x^2` (`RuleDelayed`).
- Apply rules by `Replace`, `ReplaceAll` (`/.` postfix form) or `ReplaceRepeated` (`//.` postfix form) command. *E.g.*

```
rule1=x->x^2
rule2=x_:>x^2
3 + x + x^2 + x^3 + y + y^2 + y^3;
Replace[%, rule1, {1}]                    3+ 2x^2+ x^3+ y+ y^2+ y^3
ReplaceAll[%%, rule1]                     3+ x^2+ x^4+ x^6+ y+ y^2+ y^3
Replace[%%%, rule2]                       (3+ x+ x^2+ x^3+ y+ y^2+ y^3)^2
Replace[%%%%, rule2, {1}]                 9+ x^2+ x^4+ x^6+ y^2+ y^4+ y^6
```

Rule vs. RuleDelayed, Set vs. SetDelayed

- Note the difference in the assignments:

```
f1=RandomReal[]; (*Set[f1,RandomReal[]]*)  ?f1
f2:=RandomReal[]; (*SetDelayed[f2,RandomReal[]]*) ?f2
{f1,f1,f2,f2}                                    {0.970279, 0.970279, 0.931431, 0.0333621}
```

- ... and in the rules (`rule` – **`Rule`**, `rule2` – **`RuleDelayed`**):

```
Clear[p, a];
a = 5;
rule1 = a_^3 -> a;
rule2 = a_^3 :> a;
powList = Table[p^i, {i, 4}]                {p, p^2, p^3, p^4}
powList /. rule1                            {p, p^2, 5, p^4}
powList /. rule2                            {p, p^2, p, p^4}
{rule1, rule2}                              {a_^3 -> 5, a_^3 :> a}
```

- **`SetDelayed`** and **`RuleDelayed`** evaluate when called/used, whereas ordinary **`Set`** and **`Rule`** evaluate when defined.

Rules, excercise

- Use **`ReplaceAll`** (**`/.`**) or **`Replace`** (with a level specification) to do a transformation in the list `list1`. Take 2-element lists/vectors and replace them with their norm (**`Norm`**).
  E.g. for **`list1={{1, 2}, {3, 4}, {5, 6, 7}}`** you should get `{`$\sqrt{5}$`, 5, {5, 6, 7}}`.

```
list1 = {{1, 2}, {3, 4}, {5, 6, 7}}
...
...
...
...
...
...
```

Exercise

Putting constraints on patterns and transformation rules.

We can put a constraint on a pattern, so that it matches only if the condition is applied.

- one way by `?boolFunction`, *e.g.*

```
Clear[ff,x]; ff[x_?EvenQ]:=x/2; ff/@{1,2} → {ff[1], 1}
```

  note that the following will not work

```
Clear[ff,x]; ff[x_?EvenQ[x]]:=x/2; ff/@{1,2} → {ff[1], ff[2]}
```

- another way by `/;resultOfABoolFuncion`, *e.g.*

```
Clear[gg,x]; gg[x_/;EvenQ[x]]:=x/2; gg/@{3,4} → {gg[3], 2}
```

  note that the following will not work

```
Clear[gg,x]; gg[x_/;EvenQ]:=x/2; gg/@{3,4} → {gg[3], gg[4]}
```

- By the second way, we can also constrain rules and definitions, *e.g.*

```
rule=Times[a_,b_]:>a Sin[b]/;NumericQ[a];
x+2z^3 + x z/.rule                                  x + x z + 2 Sin[z^3]
Clear[f2, f3]; f2[x_/;x^2>5]:=x^3; f3[x_]:=x^3/;x^2>5;
{f2[2],f2[3],f3[2],f3[3]}                           {f2[2], 27, f3[2], 27}
```

Putting constraints on patterns and transformation rules – examples.

- Consider vector `vec=Table[RandomInteger[{-3,3}],10];`. Define and apply a `rule` that takes any negative number in `vec` and transforms it to its square, *i.e.* if $x < 0$ return x^2 otherwise do nothing.

```
rule = ...
...
```

- Consider a list of vectors `lst`. Define and use a `rule2` that takes a list of numbers, and if the number of elements in the list is more than 3, transforms the list by taking just the first three elements from it. Use `Replace` with a level specification; `ReplaceAll` will not work.

```
lst={Range[4],RandomInteger[{0,4},3],Table[0,5], {1,3}}
rule2 = ...
...
```

- Consider a list of triplets. Select such triplets that are ordered. Implement 2 solutions, one with `Cases`, the second with `Replace` or `ReplaceAll`. Other useful commands: `OrderedQ`, `NumericQ`, `Nothing`.

```
Subsets[{-2, 3, 6, 4}, {3}]                {{-2,3,6},{-2,3,4},{-2,6,4},{3,6,4}}
Cases[%,...]                               {{-2,3,6}, {-2,3,4}}
%% /. ...                                  {{-2,3,6}, {-2,3,4}}
Replace[%%%, ..., {1}]                     {{-2,3,6}, {-2,3,4}}
```

Exercise

Cases, Count, Position, Select

- **`Cases[expr,form]`** – gives a list of the elements that match the pattern,
- **`Cases[expr,pattern->rhs]`** – gives a list of the values or `rhs` corresponding to the elements that match the pattern.
- **`Count[expr,pattern]`** – gives the number of elements in expr that match the pattern.
- **`Position[expr,pattern]`** – give a list of postions at which objects matching pattern appear in expr.
- **`Select[list, crit]`** – returns all elements for which crit$[e_i]$ evaluates as `True`.
- Examples:

```
Cases[{{1,2},{2},{3,4,1},{5,4},{3,3}}, {_,_}]          {{1,2},{5,4},{3,3}}
Count[5x^5+3 y^5,5,Infinity]                            3
Position[{1 + x^2, 5, x^4, a + (1 + x^2)^2}, x^_,2]     {{1, 2}, {3}}
Select[Range[10],IntegerQ[Sqrt[#]]&]                    {1,4,9}
```

Cases, Count, Position, Select

- Consider an $N_{\max} \times 2$ matrix `matA`. 1)Choose rows from the matrix such that *first element* − *second element* < 3. Do it twice, once with `Cases` and then with `Select`. 2) count number of rows that start with 7 or 9

```
SeedRandom[2024]; nMax = 5;
matA = RandomInteger[{0, 10}, {nMax, 2}]
Cases ...
Select ...
Count ...
```

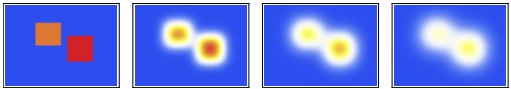
Exercise

Example. A discrete model of the heat flow

- Continous time heat equation[3] (in 2D): $\frac{\partial u(x,y,t)}{\partial t} = \alpha \left(\frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2} \right)$.

  Discrete equivalent: $\frac{u_{i,j,t+\Delta t} - u_{i,j,t}}{\Delta t} = \frac{\alpha}{h^2} \left(u_{i+1,j,t} + u_{i-1,j,t} + u_{i,j+1,t} + u_{i,j-1,t} - 4u_{i,j,t} \right)$

```
nPtsX = 70; nPtsY = 50;
pos = {{273 + 65,{20,35},{12,25}}, {273 + 100,{40,55},{20,35}}};
tempFun[i_,j_]:=Piecewise@(pos /. {c_ /; NumberQ[c], i1_, i2_} :> {c,
  IntervalMemberQ[Interval@i1, j] && IntervalMemberQ[Interval@i2, i]});
array=Table[tempFun[i, j],{i,nPtsY},{j,nPtsX}];
oneIter[arr_, al_:0.25]:=Module[{nR=Length@arr,nC=Length@First@arr,auxArr},
  auxArr = arr + al ((arr[[2;;nR, ;;]]~Join~{Table[0,nC]}) + ({Table[0,nC]}
  ~Join~arr[[1;;nR-1, ;;]]) + Join[arr[[;;, 2;;nC]], Table[{0}, nR], 2] +
  Join[Table[{0},nR], arr[[;;, 1;;nC-1]], 2] - 4arr); auxArr];
states = NestList[oneIter, array, 250];
Animate[MatrixPlot[states[[i]], Mesh->All,
  ColorFunction->(ColorData["TemperatureMap"][#1/373]&),
  ColorFunctionScaling->False],{i,Range[Length@states]}]
```



[3] https://en.wikipedia.org/wiki/Heat_equation

Questions?

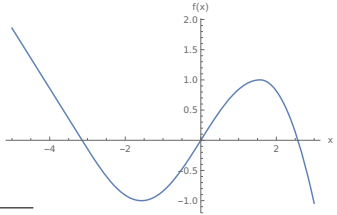
A8B17CAS
lukacjo1@fel.cvut.cz

December 2
Winter semester 2024/25

Voluntary homework

- Implement Euclidean algorithm for computing greatest common divisor[5] using a rule/rules. Use **`ReplaceRepeated`** (`//.`). Assume input to be a two-element list, *e.g.* `{20,15}`.
- Define piece-wise function $f(x)$, A) using conditioned patterns, B) using command **`Piecewise`**. Plot the function.

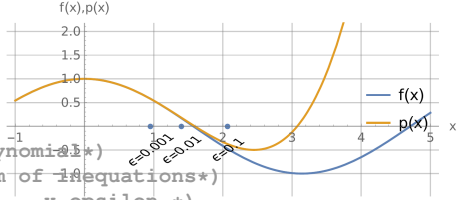
$$f(x) = \begin{cases} -(x+\pi), & x < -\pi \\ \sin(x), & -\pi \le x < \frac{\pi}{2} \\ 1 - \left(x - \frac{\pi}{2}\right)^2, & \frac{\pi}{2} \le x. \end{cases}$$



[5] https://en.wikipedia.org/wiki/Euclidean_algorithm

Voluntary homework (2)

- Approximate a function $f(x)$ by a Taylor series polynomial $p(x)$ of order n. Decide how good is the approximation for selected ε – *i.e.* find x_ε such that $|f(x) - p(x)| < x_\varepsilon$, for all $x : 0 \leq x \leq x_\varepsilon$. In the figure below, $f(x) = \cos(x)$ and $n = 4$.




```
Clear[n, x, epsRng, expr]; n = 4;
epsRng = 10.0^{-1, -2, -3}
expr = Cos[x]
approxPol = Series ... // Normal (*approximate polynomial*)
inequationsSols = Reduce ... epsRng (*reduce system of inequations*)
xEps = ... (* take last value from inequationsSols -- x_epsilon *)
Transpose@{xEps, ConstantArray[0, Length@%]}
lstPlt = ListPlot[%];
labels = (Text["\[Epsilon]=" <> ToString@#[[1]], {#[[2]], -0.5}]) & /@
    Transpose[{epsRng, xEps}];
plt = Plot[{expr, approxPol}, {x,-1,5}, PlotLegends -> Placed[{"f(x)", "p(x)"},
    {Right, Center}]];
Show[lstPlt, plt, Graphics[Rotate[#, 45 Degree] & /@
    labels],PlotRange->{{-1,5},{-1.3,2}},AxesOrigin -> {0, 0},GridLines ->
    Automatic,AxesLabel -> {"x", "f(x),p(x)"}, AspectRatio -> 2/5]
```

Voluntary homework (3)

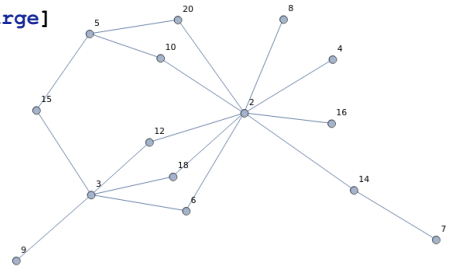
- Generate a list of pairs that represent an edge in a graph (use **UndirectedEdge**) and show it as a graph (use **Graph**). Each edge represents the relation *is my factor*. *E.g.* $N = 12 = 2^2 \cdot 3^1$, so there will be an edge $(12, 2)$ and $(12, 3)$. Omit self-edges (n, n). Useful functions/commands: **UndirectedEdge**, **Complement**, **FactorInteger**.

```
nMax = 20;
Table[ ...  , {n, 2, nMax}]
% // Flatten (*make the previous list flat*)
Graph[%, VertexLabels -> "Name", ImageSize -> Large]
```

```
Out[•]= {{}, {}, {4 ↔ 2}, {}, {6 ↔ 2, 6 ↔ 3}, {}, {8 ↔ 2}, {9 ↔ 3},
  {10 ↔ 2, 10 ↔ 5}, {}, {12 ↔ 2, 12 ↔ 3}, {}, {14 ↔ 2, 14 ↔ 7},
  {15 ↔ 3, 15 ↔ 5}, {16 ↔ 2}, {}, {18 ↔ 2, 18 ↔ 3}, {}, {20 ↔ 2, 20 ↔ 5}}

Out[•]= {4 ↔ 2, 6 ↔ 2, 6 ↔ 3, 8 ↔ 2, 9 ↔ 3, 10 ↔ 2, 10 ↔ 5, 12 ↔ 2, 12 ↔ 3,
  14 ↔ 2, 14 ↔ 7, 15 ↔ 3, 15 ↔ 5, 16 ↔ 2, 18 ↔ 2, 18 ↔ 3, 20 ↔ 2, 20 ↔ 5}
```

Out[•]=



Voluntary homework (4)

- Assume you have a matrix of numbers. Delete all columns that fulfill condition: $\max(\text{col}_i) - \min(\text{col}_i) > 4$. Useful functions/commands: `DeleteCases`, or `Select`

```
SeedRandom[2024]
nMax = 3;
RandomInteger[{-10, 10}, {nMax, nMax}]
% // MatrixForm
...
% // MatrixForm
```

$$\begin{pmatrix} 5 & 5 & -5 \\ -7 & 0 & 7 \\ 4 & -4 & 4 \end{pmatrix} \to \begin{pmatrix} 5 \\ 0 \\ -4 \end{pmatrix}$$

- Assume you have a matrix of numbers (from the previous task). Replace all columns that contain number 5 with columns of -11. Useful functions/commands: `MemberQ`, `ConstantArray`, `Length`, `ReplaceAll` (`/.`)

$$\begin{pmatrix} 5 & 5 & -5 \\ -7 & 0 & 7 \\ 4 & -4 & 4 \end{pmatrix} \to \begin{pmatrix} -11 & -11 & -5 \\ -11 & -11 & 7 \\ -11 & -11 & 4 \end{pmatrix}$$