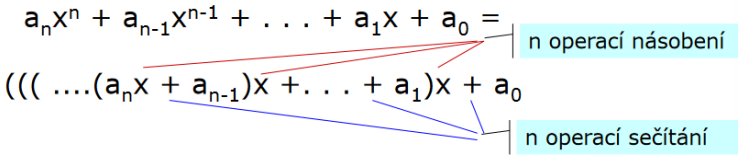
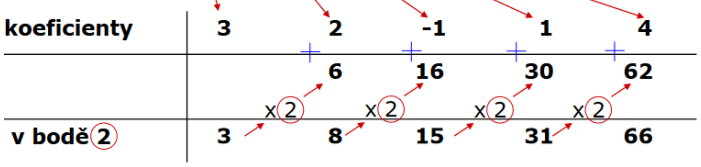
Algoritmy pro numerické výpočty

- Zápis hodnot ve **float datové struktuře**
  - 1 bit na +-
  - Mantissa – číslo
  - Exponent ($^{1,2,-1,-2}$)
  - Nemůžeme efektivně reprezentovat nějaké čísla – např. 0.1
  - Proto vzniká odchylka, se kterou musíme počítat

- **Výpočet hodnoty polynomu**
  - Definice polynomu
    - Nechť n je přirozené číslo a nechť a_0, a_1, …, a_n, jsou reálná, resp. komplexní čísla. Funkce P(x), kterou lze definovat pro všechna reálná, resp. komplexní čísla s předpisem:
    - $P(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$, (1)
    - se nazývá **mnohočlen** (jedné proměnné x s reálnými, resp. komplexními koeficienty). Místo mnohočlen se také říká **polynom** nebo **celá racionální funkce**. Čísla a_0, a_1, …, a_n, se nazývají **koeficienty polynomu** P(x).
  - **Výpočet hodnoty polynomu**
    - **Stupněm polynomu** P(x) nazýváme nejvyšší mocninu proměnné x ve výrazu (1), u níž je nenulový koeficient. Je-li v (1) $a_n \neq 0$, pak P(x) je n-tého stupně
    - Příklad: $P(x) = 3x^4 + 2x^3 - x^2 + x + 4 \Rightarrow n = 4$
    - **Výpočet hodnoty polynomu v bodě x**
      - Prostá implementace funkce vyžaduje přímý výpočet pomocí funkce, která počítá xn. Tento přístup potřebuje kvadratický čas O(n2).
      - Počet operací násobení: n + n-1 + … + 1 => O(n2)
    - **Hornerovo schéma**



Příklad: P(x) = $3x^4 + 2x^3 - x^2 + x + 4$ = $(((3x + 2)x - 1)x + 1)x + 4$
P(2) = 66



- Následující funkce předpokládá, že polynom je v paměti počítače reprezentován strukturou, která obsahuje stupeň polynomu a pole koeficientů polynomu. Funkce vrací hodnotu polynomu v bodě daném parametrem x.

```
#include <stdio.h>
#define N 4
typedef struct poly
{
  int degree;          // stupeň polynomu
  double coef[N + 1]; // koeficienty polynomu
}Tpoly;
```

```
double evalHorner (Tpoly *polynom, double x)
{
  double sum = 0.0;
  for (int i = 0; i <= polynom->degree; i++)
    sum = sum * x + polynom->coef[i];
  return sum;
}
```

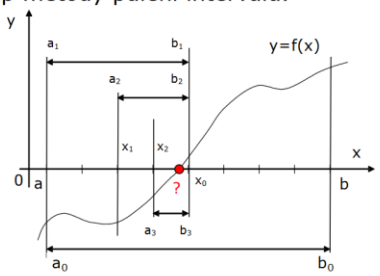
- vyčíslení hodnoty čísla zapsaného v obecné číselné soustavě:
  - číslice vstupního čísla figurují jako koeficienty polynomu
  - základ číselné soustavy jako bod, ve kterém se má polynom spočítat.

Příklad: vyčíslení hodnoty čísla

$$(2352)_6 = 2\times6^3 + 3\times6^2 + 5\times6^1 + 2\times6^0 = ((2\times6+3)\times6+5)\times6+2 = (572)_{10}$$

- **Řešení nelineárních rovnic**
  - Formulace problému:
    - Hledání reálných kořenů rovnice P(x) = 0, kde P(x) je polynom:
      - $P(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$
    - Hledáme reálné číslo α pro které platí f(α) = 0; α je kořen rovnice f(x) = 0.
    - Při určování kořene rovnice je u některých metod požadováno, aby byl separován kořen rovnice
  - Separaci lze provést několika způsoby:
    - Můžeme např. vyšetřit průběh funkce a z funkce f(x) spočítat první a druhou derivaci.
    - Z průběhů derivací lze zjistit, ve kterých intervalech je funkce rostoucí a klesající a vyšetřit pak lokální minima a maxima.
    - Je důležité si uvědomit, že reálné kořeny rovnice jsou průsečíky grafu funkce a osy x.
    - Zjištěné intervaly potom použijeme pro přibližný výpočet kořenů.
  - **Metoda půlení intervalu (bisekce)**
    - konvergentní, univerzální metoda,
    - musí být splněny dvě podmínky:
      - 1. funkce f musí být spojitá pro $\forall x \in I_0 = < a_0, b_0 >$
      - 2. funkční hodnoty v krajních bodech zvoleného intervalu musí mít opačná znaménka tj. musí platit: $f(a_0) \times f(b_0) < 0$.
    - pokud jsou obě podmínky splněny, pak tato metoda vždy konverguje

Princip metody půlení intervalu.



- $x_i = s_i$ je střed příslušného intervalu

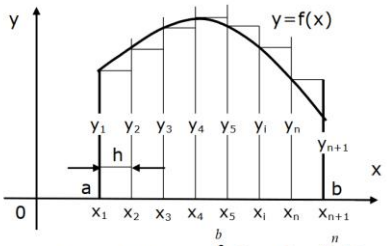
Příklad: funkce pro hledání kořene metodou půlení intervalu

```
double root_equation (double a, double b,
    double eps,double(*evalFun)(double))
{
  double middle = (a + b) / 2;
  double fmid = evalFun(middle);
```

Příklad: funkce pro hledání kořene metodou půlení intervalu - pokračování

```
  while (fabs(fmid) > eps) {
    if (evalFun(a) * fmid < 0)
      b = middle;
    else
      a = middle;
    if (fabs (fmid) > eps) {
      middle = (a + b) / 2;
      fmid = evalFun(middle);
    }
  }
  return middle;
}
```

- Numerický výpočet určitého integrálu
  - **Definice:**
    - Jestliže je funkce f(x) spojitá v uzavřeném intervalu <a,b> a známe-li její primitivní funkci F(x),
    - můžeme vypočítat určitý integrál funkce f(x) v mezích od a do b pomocí vztahu:
      - $\int_a^b f(x)dx = F(b) - F(a)$
    - Metody numerického integrování užíváme v takových případech, kdy je obtížné najít funkci F(x), nebo v případě, že funkce f(x) je dána tabulkou
  - **Obdélníková metoda**

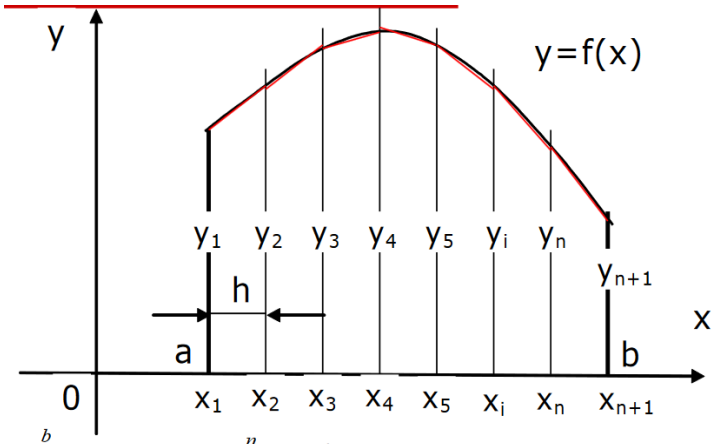


Vzorec pro výpočet: $\int_a^b f(x)dx \approx \sum_{i=1}^{n} h_i y_i = h\sum_{i=1}^{n} y_i$,

kde šířka dílčího intervalu h = (b-a)/n

Příklad: funkce pro výpočet integrálu obdélníkovou metodou

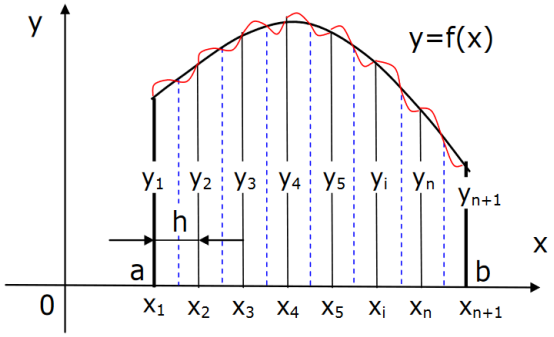
```
double integrate_rectangle (double a, double b,
                int n, double (*evalFun)(double))
{
  double step, sum = 0.0;
  step = (b - a) / n;
  for (double x = a; x < b - (step/2); x += step)
    sum += evalFun(x);
  sum *= step;
  return sum;
}
```

- **Lichoběžníková metoda**
  - Přesnější integraci funkce f(x) získáme použitím lichoběžníků, kterými aproximujeme danou funkci.
  - 

$$\int_a^b f(x)dx \approx h\sum_{i=1}^{n}\frac{y_i + y_{i+1}}{2} = h(y_1/2 + y_{n+1}/2 + \sum_{i=2}^{n} y_i)$$

Příklad: funkce pro výpočet integrálu lichoběžníkovou metodou

```
double integrate_trapezoid (double a, double b,
                 int n,double (*evalFun)(double))
{
  double step, sum = 0.0;
  step = (b - a) / n;
  for (double x = a+step; x < b-step; x += step)
    sum += evalFun(x);
  sum += (evalFun(a) + evalFun(b))/2;
  sum *= step;
  return sum;
}
```

- **Simpsonova mmetoda**
  - Počet dílčích intervalů musí být sudé číslo
  - Tři sousední body na křivce f(x) se aproximují vhodnou parabolou.
  - 

$$\int_a^b f(x)\ dx \approx \frac{h}{3}(y_1 + 4y_2 + 2y_3 + 4y_4 + 2y_5 + \ldots + 2y_{n-1} + 4y_n + y_{n+1})$$

Numerický výpočet určitého integrálu

- Experimentální výsledky:

n	obdélníková	lichoběžníková	Simpsonova
10	1.285000	1.335000	1.333333
100	1.328350	1.333350	1.333333
1000	1.332834	1.333334	1.333333
10000	1.333283	1.333333	1.333333
100000	1.333328	1.333333	1.333333
přesná	1.333333	1.333333	1.333333