Pokud chci znovu použít bod $c$, musí platit

$$d - c = t^2(b - a) \quad \text{nebo} \quad d - c = t(1 - t)(b - a).$$

První možnost vede na kvadratickou rovnici

$$t^2 - 2t + 1 = 0,$$

jejíž řešení $t = 0.5$ nevyhovuje podmínkám. Druhá možnost odpovídá rovnici

$$t^2 + t - 1 = 0, \tag{7.4}$$

jejíž řešení

$$\tau = \frac{-1 + \sqrt{5}}{2} = 0.618033988749\ldots\ , \tag{7.5}$$

nazývající se **poměr zlatého řezu**, dává této metodě název.

```
Interval MinGoldenSection(Interval I, double tol)
{
  const double t = 0.6180339887498948482045868343656 4;
  double a,b,c,d,fc,fd;
  a = I.a; b = I.b;
  c = a + (1-t)*(b-a);
  d = a + t*(b-a);
  fc = f(c); fd = f(d);
  while(b-a>tol)
  {
    if(fc<=fd)
    {
      b=d;
      d=c;fd=fc;
      c = a + (1-t)*(b-a);fc = f(c);
    }
    else
    {
      a=c;
      c=d;fc=fd;
      d = a + t*(b-a);fd = f(d);
    }
  }
  return new Interval(a,b);
}
```

### Fibonacciova metoda

Je-li dopředu znám počet kroků, je o něco efektivnější tzv. Fibonacciova metoda. Fibonacciova čísla jsou definována rekurentním vztahem

$$F_0 = 1,\ F_1 = 1;\ F_n = F_{n-1} + F_{n-2}. \tag{7.6}$$

Začátek posloupnosti vypadá takto

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 \dots$$

Označme

$$t_n = \frac{F_n}{F_{n+1}}, \; n = 1, 2, \dots.$$

Budeme tedy pracovat s posloupností

$$\frac{1}{2}, \frac{2}{3}, \frac{3}{5}, \frac{5}{8}, \frac{8}{13}, \frac{13}{21}, \frac{21}{34}, \frac{34}{55}, \frac{55}{89} \dots$$

Pro počítání prvních N poměrů Fibonacciových čísel můžeme použít následující třídu, která si uchovává Fibonacciova čísla v členské proměnné F typu pole. Rekurentní vztah 7.6 lze rovněž vyřešit. Funkce generující Fibonacciova čísla vypadá takto

$$F_n = \frac{5+\sqrt{5}}{10}\left(\frac{1+\sqrt{5}}{2}\right)^n + \frac{5-\sqrt{5}}{10}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

```
class Fibonacci
{
  public long[] F;
  int N;
  const double c1=0.27639320225002103035908263312687;
  const double c2=1-c1;
  const double r1=-0.61803398874989484820458683436564;
  const double r2=1-r1;

  public double FN(int i)
  {
    return c1*Math.Pow(r1,i)+c2*Math.Pow(r2,i);
  }

  public Fibonacci(int N)
  {
    this.N = N;
    F=new long[N+1];
    F[0]=1;
    F[1]=2;
    for(int i=2;i<=N;i++)
      F[i]=F[i-1]+F[i-2];
  }

  public double ta(int k)
  {
    return (double)(F[k])/F[k+1];
  }

  public double t(int k)
  {
```

```
    return FN(k)/FN(k+1);
  }
}
```

Platí

$$t_{n-1}t_n = \frac{F_{n-1}}{F_n}\frac{F_n}{F_{n+1}} = \frac{F_{n-1}}{F_{n+1}} = \frac{F_{n+1}-F_n}{F_{n+1}} = 1 - t_n. \tag{7.7}$$

Nechť n je známý počet kroků. Spočítáme $t = t_n$ a uděláme krok intervalové redukce 7.2. V dalším kroku zvolíme $t = t_{n-1}$ a obecně v $k$-tém kroku zvolíme $t = t_{n-k}$, poslední krok provedeme s hodnotou $t = t_1 = \frac{1}{2}$. Ze vztahu 7.7 plyne, že v $k$-tém kroku můžeme použít již dříve vypočtenou hodnotu z kroku $k-1$. Jeden krok metody nás stojí výpočet jedné funkční hodnoty, jako u metody zlatého řezu. Poslední krok vyžaduje speciální ošetření, protože body $c$ a $d$ splynou. Vezmeme bod $c+\epsilon$, kde $\epsilon$ je dostatečně malé, aby příliš nezměnilo kvalitu poslední redukce ($t_1 = \frac{1}{2}$) a současně ještě tak velké, aby šlo spolehlivě rozlišit mezi $f(c)$ a $f(c+\epsilon)$.

```
Interval MinFibonacci(Interval I, int n)
{
  const double epsilon=1E-10;
  double t=Fibonacci.t(n);
  double a,b,c,d,fc,fd;
  a = I.a; b = I.b;
  c = a + (1-t)*(b-a);
  d = a + t*(b-a);
  fc = f(c); fd = f(d);
  for(int k=1;k<n;k++)
  {
    t = Fibonacci.t(n-k);
    if(fc<=fd)
    {
      b=d;
      d=c;fd=fc;
      c = a + (1-t)*(b-a);fc = f(c);
    }
    else
    {
      a=c;
      c=d;fc=fd;
      d = a + t*(b-a);fd = f(d);
    }
  }
  d=c+epsilon;
  fd=f(d);
  if(fc<=fd)
    b=d;
  else
    a=c;
  return new Interval(a,b);
}
```

Ze vztahu 7.7 plyne

$$t_n t_{n+1} = 1 - t_{n+1} \Rightarrow t_{n+1} = \tfrac{1}{1+t_n},$$

$$t_{n+2} = \tfrac{1}{1+t_{n+1}} = \tfrac{1+t_n}{2+t_n}, \tag{7.8}$$

$$t_{n+2} - t_n = \tfrac{1+t_n}{2+t_n} - \tfrac{1+t_{n-2}}{2+t_{n-2}} = \tfrac{t_n - t_{n-2}}{(2+t_n)(2+t_{n-2})},\ (2+t_n)(2+t_{n-2}) > 0. \tag{7.9}$$

Protože je

$$\begin{aligned} t_3 - t_1 &= \frac{3}{5} - \frac{1}{2} = \frac{1}{10} > 0, \\ t_4 - t_2 &= \frac{5}{8} - \frac{2}{3} = -\frac{1}{24} < 0, \end{aligned}$$

plyne z 7.9 indukcí

$$t_{2n-1} < t_{2n+1}, \tag{7.10}$$

$$t_{2n} > t_{2n+2}. \tag{7.11}$$

Protože je zřejmě posloupnost $t_n$ ohraničená,

$$0 < t_n < 1,$$

existují konečné limity

$$\lim_{n\to\infty} t_{2n-1} = \tau_1,\ \lim_{n\to\infty} t_{2n} = \tau_2.$$

Limitním přechodem v 7.8 dostaneme

$$\tau_i = \frac{1+\tau_i}{2+\tau_i} \Leftrightarrow \tau_i^2 + \tau_i - 1 = 0.$$

Obě dvě limity proto splývají s kladným kořenem kvadratické rovnice 7.4, a tudíž

$$\lim_{n\to\infty} t_n = \tau = \frac{-1+\sqrt{5}}{2} = 0.618033988749\ldots$$

Pro velká $n$ je tedy krok Fibonacciovy metody prakticky shodný s metodou zlatého řezu. Celková redukce délky intervalu je po n krocích podle 7.7

$$t_1 t_2 \ldots t_n = \begin{cases} (1-t_2)(1-t_4)\ldots(1-t_n) & \text{pro n sudé,} \\ (1-t_2)(1-t_4)\ldots(1-t_{n-1})t_n & \text{pro n liché.} \end{cases}$$

Z 7.10 plyne

$$\begin{aligned} t_{2n} > \tau \Rightarrow 1 - t_{2n} &< 1 - \tau = \tau^2, \\ t_{2n-1} &< \tau. \end{aligned}$$

V každém případě je výsledný interval, který obdržíme Fibonacciovou metodou, o něco kratší, než ten který poskytne metoda zlatého řezu:

$$t_1 t_2 \ldots t_n = \frac{1}{F_{n+1}} < \tau^n.$$

Následující tabulka porovnává efektivitu metody zlatého řezu s Fibonacciovou metodou. Rozdíl je sice znatelný, ale z hlediska rychlosti konvergence bezvýznamný. Vyjdeme-li z intervalu délky jedna, budeme potřebovat pro dosažení délky $< 0,01$ u obou metod 10 kroků:

| $n$ | $t_1 t_2 \dots t_n$ | $\tau^n$ | $t_1 t_2 \dots t_n / \tau^n$ |
|---|---|---|---|
| 1 | 0,5 | 0,618033988749895 | 0,809016994374947 |
| 2 | 0,333333333333333 | 0,381966011250105 | 0,872677996249965 |
| 3 | 0,2 | 0,23606797749979 | 0,847213595499958 |
| 4 | 0,125 | 0,145898033750316 | 0,85676274578121 |
| 5 | 0,0769230769230769 | 0,0901699437494743 | 0,853089995673036 |
| 6 | 0,0476190476190476 | 0,0557280900008412 | 0,854489138571388 |
| 7 | 0,0294117647058823 | 0,034441853748633 | 0,853954172169077 |
| 8 | 0,0181818181818182 | 0,0212862362522082 | 0,854158432068141 |
| 9 | 0,0112359550561798 | 0,0131556174964248 | 0,854080400196588 |
| 10 | 0,00694444444444444 | 0,00813061875578336 | 0,854110204036417 |
| 15 | 0,000626174076393237 | 0,000733137435857406 | 0,854101899272031 |
| 20 | 5,64620857094461E-05 | 6,61069613518961E-05 | 0,854101966794252 |

### 7.1.2 Metody interpolace

Tyto metody nemusí vždy konvergovat. Budeme podobně jako např. u metody sečen potřebovat dobrou počáteční aproximaci minima.

**Metoda kvadratické (parabolické) interpolace** Jedná se vlastně o analogii metody sečen pro optimalizační úlohu. V okolí bodu $x_k$ nahradíme zadanou funkci kvadratickým interpolačním polynomem $P$ (polynom prvního stupně nemá minimum) a novou hodnotu $x_{k+1}$ získáme jako bod minima polynomu $P$.

Bude se nám proto hodit obecný vzorec pro minimum kvadratického interpolačního polynomu. Uvažujme proto následující interpolační úlohu:

| $i$ | 0 | 1 | 2 |
|---|---|---|---|
| $x_i$ | $a$ | $b$ | $c$ |
| $y_i$ | $f(a)$ | $f(b)$ | $f(c)$ |

Interpolační polynom můžeme psát ve tvaru

$$P = f(b) + c_1(x-b) + c_2(x-b)^2.$$

Podmínky interpolace

$$\begin{aligned} P(b) &= f(b), \\ P(a) &= f(b) + c_1(a-b) + c_2(a-b)^2 = f(a), \\ P(c) &= f(b) + c_1(c-b) + c_2(c-b)^2 = f(c), \end{aligned}$$