Numerické výpočty-II

Jiří Zelinka

Obsah

KAPITOLA 1

Polynomy

Nebudeme se zda zabývat příliš teorií a základními pojmy jako např. stupeň polynomu apod. Literatura v tomto směru je velmi obsáhlá a čtenáři jistě nebude činit pražádné potíže si nějakou vyhledat. Nás budou zajímat především výpočty.

První věc, na kterou bych rád upozornil, je nejednostnost zápisu polynomů v různé literatuře. Zpravidla se setkáme se zápisem

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

ale občas narazíme na opačné pořadí indexů, tedy na polynom ve tvaru

$$P(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n.$$

Pro konkrétní polynom, kdy keoficienty nejsou vyjádřeny obecně ale čísly, je to samozřejmě jedno, ale pokud chceme pro polynom použít nějaké tvrzení, je potřeba si dávat pozor, v jakém pořadí ta či ona věta uvádí pořadí koeficientů. V této příručce se budu držet prvního způsobu, tedy indexy u koeficientů budou stejné jako příslušná mocnina x. Pokud bude potřeba zvýraznit stupeň polynomu, budeme používat značení P_n.

1. Hornerovo schema a základní úkony s polynomy

Na polynom budeme nahlížet jako na funkci – vrazíte do ní x a vypadne funkční hodnota podle daného vztahu. Základní věc, kterou tedy s polynomem potřebujeme dělat, je výpočet funkční hodnoty. Nejrychlejší metodou k tomu je tzv. Hornerovo schema. Nebudu uvádět jeho podrobné odvození, které by pro čtenáře bylo snadným cvičením. Hornerovo schema je založeno na dělení polynomu P polynomem prvního stupně $P_1(x) = x - c$, tedy

$$P(x) = (x - c) \cdot Q(x) + A, \tag{1}$$

kde Q je podíl (polynom stupně o jedna menší než P) a A je zbytek po dělení, který musí mít stupěň menší než dělitel, tedy je to konstanta. Z uvedeného zápisu je jasné, že $P(c) = A$, tedy tento zbytek po dělení je současně hodnota polynomu P v bodě c. Z (1) se porovnáním koeficientů u stejných mocnín dají odvodit vztahy pro koeficienty polynomu

Q, Předpokládáme-li polynom Q ve tvaru

$$Q(x) = b_{n-1}x^{n-1} + \cdots + b_1 x + b_0$$

dostáváme postupně

$$\begin{aligned} b_{n-1} &= a_n \\ b_{n-2} &= a_{n-1} + c \cdot b_{n-1} \\ &\vdots \\ b_{k-1} &= a_k + c \cdot b_k \\ &\vdots \\ b_0 &= a_1 + c \cdot b_1 \\ A &= a_0 + c \cdot b_0. \end{aligned}$$

Hornerovo schema zpravidla v literatuře nalezname coby následujíccí tabulku:

	a_n	a_{n-1}	a_{n-2}	$\cdots$	a_2	a_1	a_0
c	b_{n-1}	b_{n-2}	b_{n-3}	$\cdots$	b_1	b_0	A

Pravidlo pro zapamatování výpočetního algoritmu je: „*První čisIo opíšeme ($b_{n-1} = a_n$), každé další dostaneme tak, že k číslu nad čarou připočteme c násobek předchozího čísla ($b_{k-1} = a_k + c \cdot b_k$)*“.

Hornerovo schema se při ručních výpočtech nejčastěji používá k testování, zda dané číslo c je kořenem polynomu, v tom případě vyjde $A = 0$. Ale vidíme, že kromě hodnoty polynomu v bodě c dostáváme i podíl – polynom Q.

Někoho možná napadne, co by se stalo, kdybychom Hornerovo schema použili se stejnou hodnotou c znovu, tentokrát na polynom Q, pak znovu na výsledný podíl a tak dál. Pro tento účel si označíme polynom Q jako Q_1 a hodnotu A jakožto A_0, v dalším kroku dostaneme podíl Q_2 a hodnotu A_1, a tak dál, takže obecně máme

$$Q_k(x) = (x - c) \cdot Q_{k+1}(x) + A_k.$$

Hornerovo schema pak (symbolicky zkráceno) vypadá takto:

	P			
c		Q_1		A_0
c		Q_2	A_1	
c	Q_3	A_2		
$\vdots$		$\iddots$		
c	A_n			

Pro polynom P pak dostáváme

$$\begin{aligned} P(x) &= (x-c)Q_1(x) + A_0 = \\ &= (x-c)((x-c)Q_2(x) + A_1) + A_0 = \\ &= (x-c)^2 Q_2(x) + A_1(x-c) + A_0 = \\ &= (x-c)^2((x-c)Q_3(x) + A_2) + A_1(x-c) + A_0 = \\ &= (x-c)^3 Q_3(x) + A_2(x-c)^2 + A_1(x-c) + A_0 = \cdots = \\ &= A_n(x-c)^n + A_{n-1}(x-c)^{n-1} + \cdots + A_1(x-c) + A_0 \end{aligned}$$

Hodnoty $A_n, \ldots, A_0$ jsou tedy koeficienty polynomu P posunutého do bodu c. Toto vyjádření se dá také velmi dobře použít pro výpočet derivací polynomu P v bodě c, ale to bychom předbíhali.

V Matlabu definujeme polynom pomocí vektoru jeho koeficientů od nejvyšší mocniny, takže příkazem

```
>> P=[1 3 -2 0 5]
```

definujeme polynom $P(x) = x^4 + 3x^3 - 2x^2 + 5$, jak se dá snadno ověřit převodem polynomu na smybolický objekt:

```
>> poly2sym(P)
ans =
x^4 + 3*x^3 - 2*x^2 + 5
```

Pro výpočet hodnoty polynomu v bodě použijeme funkci `polyval` a můžeme ji apikovat nejen na jednu hodnotu ale na vektor či matici hodnot, přičemž hodnoty polynomu se počítají pro každou sloužku zvlášť. Takže graf polynomu na intervalu můžeme získat třeba pomocí příkazů

```
>> P=[1 3 -2 0 5];
>> x=-3:0.01:3;
>> y=polyval(P,x);
>> plot(x,y)
```

Operace s polynomy. Mezi základní úkony, které lze s polynomy provádět, patří sčítání, násobení skalárem, vzájemné násobení a dělení se zbytek. První dvě operace patrně nepotřebují další komentáře, jen je potřeba vědět, že pro sčítání polynomů v Matlabu musí mít příslušné vektory koeficientů stejnou délku, takže je nutné je doplnit v případě potřeby nulami. Pro násobení polynomů se používá funkce `conv`, kde jako vstupní parametry zadáme polynomy, které chceme násobit.

Dělení polynomu se zbytkem se provádí podobně, jako je tomu u celých čísel. Nejznámějším použitím dělení polynomu se zbytkem je Eukleidův algoritmus pro hledání největšího společného dělitele dvou polynomů. Algerotmus je všeobecně známý, proto jej zde nebudeme uvádět. Pro dělení se zbytkem v matlabu je možné použít funkci `deconv`, která

má dva vstupní a dva výstupní parametry. Ná výstupu dostáváme podíl a zbytek po dělení. Například při dělení polynomu x^4+3x^3-4x+1 polynomem x^2+1 dostáváme

```
>> P=[1 3 0 -4 1];
>> Q=[1 0 1];
>> [D,R]=deconv(P,Q)
D =
     1     3    -1
R =
     0     0     0    -7     2
```

Zbytek po dělení `R` má formálně stejný stupeň jako dělenec `P`, aby platila rovnost `P=D*Q+R`. Tuhle skutečnost je potřba brát v potaz při některých výpočtech v Matlabu, například právě u zmíněného Eukledova algoritmu.

1.1. Derivace polynomu. V této části trochu předběhneme učivo matematického kursu, protože derivace polynomu se nám bude hodit v dalším výkladu a navíc pro polynomy se počítá poměrně jednoduše. Obecně lze říci, že derivace funkce udává, jak rychle se funkce měmní. Tedy pokud funkce hodně rychle roste, má derivace velké hodnoty, pokud se funkce na nějakém intervalu nemění (je konstantní), je tam její derivace nulová a tak podobně. Derivace je kladná tam, kde funkce roste, záporná, když funkce klesá, v bodech, kde má funkce lokální minimum nebo maximum, je derivace nulová. Pro polynom

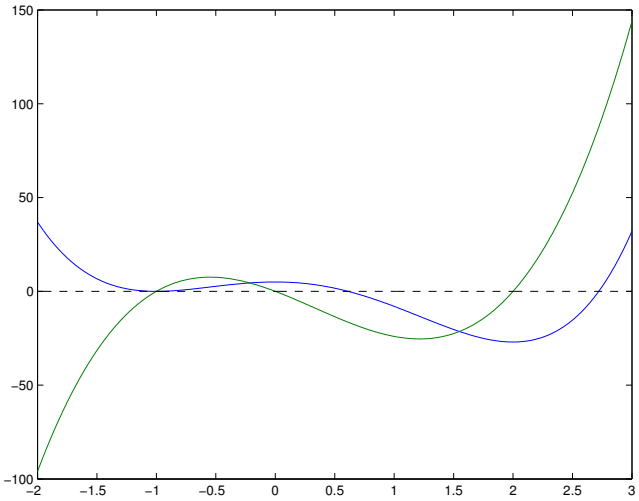
$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

se jeho derivace vypočítá podle vztahu

$$P'(x) = n \cdot a_n x^{n-1} + (n-1) \cdot a_{n-1}x^{n-2} + \cdots + 2 \cdot a_2 x + a_1$$

V Matlabu se derivace vypočítá příkazem `polyder`, takže pokud si chceme zobrazit polynom spolu s jeho derivací, můžeme to udělat například takto:

```
>> P=[3 -4 -12 0 5];
>> DP=polyder(P)
DP =
    12   -12   -24     0
>> x=-2:0.01:3;
>> y=polyval(P,x);
>> dy=polyval(DP,x);
>> z=zeros(size(x));
>> plot(x,y,x,dy,x,z,'k--')
```



Modře je zobrazen polynom, zeleně jeho derivace.

2. Kořeny polynomů

Najít kořen daného polynomu je jednou ze základních matematických úloh. K ověřování, zda dané číslo je nebo není kořenem polynomu zpravidla používáme Hornerovo schema, existují ale další nástroje, které nám mohou hledání kořenů usnadnit.

Například pokud máme polynom s celočíselnými koeficienty a zajímají nás racionální kořeny, tedy kořeny ve tvaru $\frac{p}{q}$, kde p je celé číslo, q je přirozené číslo a p a q jsou nesoudělná, tak se dá lehce zjistit, že koeficient u nejvyšší mocniny polynomu (tedy a_n) musí být dělitelný číslem q, zatímco absolutní člen (t.j. a_0) musí být dělitelný p.

Další pomůckou může být stanovení oblasti, kde všechny kořeny leží. K tomu nám může pomoci například tvrzení, které lze najít i s důkazem v [**Horová and Zelinka, 2008**].

Nechť

$$\begin{aligned} A &= \max\left(|a_0|,\ldots,|a_{n-1}|\right), \\ B &= \max\left(|a_1|,\ldots,|a_n|\right), \end{aligned}$$

kde a_k, $k = 0,1,\ldots,n$, $a_0 a_n \neq 0$, *jsou koeficienty polynomu* P, *Pak pro všechny kořeny* $x_k, k = 0,1,\ldots,n$, *polynomu* P *platí*

$$\frac{1}{1+\dfrac{B}{|a_0|}} \leq |x_k| \leq 1 + \frac{A}{|a_n|}.$$

Všechny kořeny tedy v komplexním oboru leží v mezikruží, jehož hranice jsou dány uvedenou nerovností. Uvedené hranice, zejména pak

horní hranice, jsou poměrně hodně hrubě odhadnuty. Například pro polynom, jehož kořeny jsou čísla od jedné do šesti, dostaneme následující hranice:

```
>> P=poly(1:6)
P =
  Columns 1 through 6
      1    -21    175   -735    1624     -1764
  Column 7
      720
>> n=length(P);
>> A=max(abs(P(2:n)))
A =
        1764
>> B=max(abs(P(1:n-1)))
B =
        1764
>> H=1+A/P(1) % horni hranice
H =
        1765
>> D=1/(1+B/P(1)) % dolni hranice
D =
   5.6657e-04
```

Existují další kriteria, která v některých případech mohou poskytnout lepší odhad velikosti absolutní hodnoty kořenů, uveďme aspoň některá: *Pro všechny kořeny* $x_k, k = 0, 1, \ldots, n$, *polynomu* P *platí*

$$\begin{aligned} |x_k| &\leq \max\left\{1, \sum_{j=0}^{n-1}\left|\frac{a_j}{a_n}\right|\right\} \\ |x_k| &\leq 2\max\left\{\left|\frac{a_{n-1}}{a_n}\right|, \sqrt{\left|\frac{a_{n-2}}{a_n}\right|}, \ldots, \sqrt[n]{\left|\frac{a_0}{a_n}\right|}\right\} \\ |x_k| &\leq \max\left\{\left|\frac{a_0}{a_n}\right|, 1+\left|\frac{a_1}{a_n}\right|, \ldots, 1+\left|\frac{a_{n-1}}{a_n}\right|\right\}. \end{aligned}$$

Dalším zjednodušením při hledání kořenů polynomu může být odstranění násobných kořenů. Obecně platí, že každý polynom lze zapsat ve tvaru

$$P(x) = a_n(x - x_1)^{n_1} \cdots (x - x_k)^{n_k},$$

kde $x_1,\ldots,x_k$ jsou vzájemně různá komplexní čísla, $n_1,\ldots,n_k$ jsou jejich násobnosti a $n_1 + \cdots + n_k = n = st(P)$. Platí také, že pokud je kořen

x_i násobnosti $n_i > 1$, pak tento kořen je i kořenem derivace polynomu P s násobností $n_i - 1$. Odtud plyne, že snížit násobnost všech kořenů na 1 lze tak, že určíme nějvětší společný dělitel D polynomu P a jeho derivace P'. Kořeny tohoto nějvětšího společného dělitele jsou právě ty kořeny původního polynomu, které mají násobnost větší než jedna a jejich násobnost v děliteli D je o jedna měnší než v polynomu P. Vydělením polynomu P dělitelem D získáme tedy polynom, který má stejné kořeny jako P, ale všechny jsou násobnosti 1.

Z numerického hlediska je potřeba poznamenat, že tento postup v praxi funguje dobře pro polynomy se celočíselnými koeficienty, které se pohybují v „rozumných“ mezích. Ale i v jednoduchých případech je potřeba postupovat obezřetně:

```
>> format long
>> P=poly([1 1 1 1])
P =
     1    -4     6    -4     1
>> DP=polyder(P)
DP =
     4   -12    12    -4
>> roots(P)
ans =
  1.000217162530750
  0.999999969335793 + 0.000217131857745i
  0.999999969335793 - 0.000217131857745i
  0.999782898797672
>> roots(DP)
ans =
  1.000004930958320 + 0.000008540746793i
  1.000004930958320 - 0.000008540746793i
  0.999990138083364
```

Vidíme, že pokud bychom posuzovali shodnost kořenů u uvedeného polynomu a jeho derivace, přičemž ani nepožadujeme příliš velkou přesnost, tak nejenže ke shodě nedochází, ale kořeny ani nejsou násobné. Nicméně pokud bychom hledali největší společný dělitel, postup povede ke správnému výsledku:

```
>> [Q,R]=deconv(P,DP)
Q =
   0.250000000000000  -0.250000000000000
R =
     0     0     0     0     0
```

```
>> D=Q % D je nejv.spol.delitel
D =
   0.250000000000000  -0.250000000000000
>> roots(D)
ans =
     1
```

Vidíme, že výsledek je přesný, a to i navzdory numerickým nepřesnostem během výpočtu. Ukažme si ještě jeden příklad, kde budou dva násobné kořeny blízko sebe:

```
>> P=poly([0.9 0.9 1.1 1.1 1.1])
P =
 1.0000 -5.1000  10.3800  -10.5380  5.3361 -1.0781
>> DP=polyder(P)
DP =
 5.0000 -20.4000  31.1400 -21.0760  5.3361
>> [Q1,R1]=deconv(P,DP)
Q1 =
  0.2000 -0.2040
R1 =
    0    0   -0.0096    0.0298   -0.0306    0.0105
>> R1(1:2)=[] % odstranime pocatecni nuly
R1 =
   -0.0096    0.0298   -0.0306    0.0105
>> [Q2,R2]=deconv(DP,R1)
Q2 =
 -520.8333  510.4167
R2 =
   1.0e-11 *
         0         0   -0.1766    0.2515   -0.0769
```

V tomto místě je nutné usoudit, že zbytek po dělení je ve skutečnosti nulový, tudíž největší společný dělitel je předchozí zbytek, tedy polynom $R1$.

```
>> D=R1
D =
   -0.0096    0.0298   -0.0306    0.0105
>> P0=deconv(P,D)
P0 =
 -104.1667  208.3333 -103.1250
>> roots(P0)
ans =
```

```
    1.1000
    0.9000
>> format long
>> roots(P0)
ans =
   1.100000000001784
   0.899999999998279
```

Vidíme, že výsledek je opět poměrně přesný. Kořeny původního polynomu Matlab spočítá s daleko větší chybou:

```
>> roots(P)
ans =
  1.100021372535598 + 0.000037011183255i
  1.100021372535598 - 0.000037011183255i
  1.099957254925198
  0.900000434848828
  0.899999565154781
```

2.1. Separace reálných kořenů. Ukážeme si algoritmus, s jehož pomocí je možné přesně určit počet reálných polynomů v daném intervalu, pokud předpokládáme, že všechny reálné kořeny jsou jednoduché. Tento algoritmus je založen na kostrukci tzv. Sturmovy posloupnosti, která se definuje následovně:

Posloupnost reálných polynomů

$$P = P_0, P_1, \ldots, P_m$$

se nazývá Sturmovou posloupností *příslušnou polynomu P, jestliže*

(1) *Všechny reálné kořeny polynomu P_0 jsou jednoduché.*
(2) *Je-li x_0 reálný kořen polynomu P_0, pak $signP_1(x_0) = -signP_0'(x_0)$.*
(3) *Jestliže x_0 je reálný kořen polynomu P_i, platí*

$$P_{i+1}(x_0)P_{i-1}(x_0) < 0,$$

pro $i = 1, 2, \ldots, m-1$.
(4) *Poslední polynom P_m nemá reálné kořeny.*

Sturmovu posloupnost lze zkonstruovat poměrně snadno, Pokud předpokládáme, že výchozí polynom $P = P_0$ nemá reálné kořeny, stačí položit $P_1 = -P_0'$, abychom zaručili splnění druhé vlastnosti. Třetí vlastnost z definice dostaneme, pokud polynom P_{i+1} vezmeme jako záporně vzatý zbytek po dělení $P_{i-1} : P_i$,tedy

$$P_{i-1} = P_i \cdot Q - P_{i+1}$$

Pro kořen x_0 polynomu P_i tedy máme $P_{i-1}(x_0) = -P_{i+1}(x_0)$, přičemž daný výraz nemůže být nulový, jinak bychom indukcí dostali, že x_0 je kořenem P_0 i P_1, což je spor s tím, že kořeny P_0 jsou jednoduché.

Konstrukce založená na dělení se zbytkem zaručuje, že stupně polynomů v posloupnosti postupně klesají. Poslední polynom P_m je zpravidla konstatní, ale je možné konstrukci ukončit i dříve, pokud víme, že polynom nemá reálné kořeny, např. pro polynom $x^2 + 1$.

Počet kořenů v daném intervalu pak lze zjistit pomocí Sturmovy věty:

Počet reálných kořenů polynomu P v intervalu $[a, b]$ je roven $W(b) - W(a)$, kde $W(x)$ je počet znaménkových změn ve Sturmově posloupnosti $P_0(x), \ldots, P_m(x)$ v bodě x (z níž jsou vyškrtnuty nuly).

Předpokládám, že pojem *počet znaménkových změn* je natolik intiutivně jasný, že nepotřebuje definici. Důkaz Sturmovy měty je založen na faktu, že k navýšení či snížení počtu znaménkových změn může dojít jen při přechodu přes kořen některého z polynomů ve Sturmově posloupnosti. Ze druhé vlastnosti v definici Sturmovy posloupnosti skutečně plyne, že pokud x roste, pak při přechodu přes kořen $P = P_0$ jsou dvě možnosti: buď přecházíme ze záporných hodnot polynomu do kladných, v tom případě polynom P_0 roste, jeho derivace je tedy v okolí kořene kladná, tudíž P_1 je tam záporný a přibude jedna znaménková změna. Nebo P_0 v kořenu klesá, takže P_1 je kladný a také přibude jedna znaménková změna.

	$x-h$	x	$x+h$
P_0	$-$	0	$+$
P_1	$-$	$-$	$-$
$W(x)$	0	0	1

	$x-h$	x	$x+h$
P_0	$+$	0	$-$
P_1	$+$	$+$	$+$
$W(x)$	0	0	1

Při přechodu přes kořen polynomu P_i pro $i > 0$ jsou celkem čtyři možnosti, při žádné z nich však podle třetí vlstnosti z definice nedochází ke změně počtu znaménkových změn:

	$x-h$	x	$x+h$
P_{i-1}	$-$	$-$	$-$
P_i	$-$	0	$+$
P_{i+1}	$+$	$+$	$+$
$W(x)$	1	1	1

	$x-h$	x	$x+h$
P_{i-1}	$+$	$+$	$+$
P_i	$-$	0	$+$
P_{i+1}	$-$	$-$	$-$
$W(x)$	1	1	1

	$x-h$	x	$x+h$
P_{i-1}	$-$	$-$	$-$
P_i	$+$	0	$-$
P_{i+1}	$+$	$+$	$+$
$W(x)$	1	1	1

	$x-h$	x	$x+h$
P_{i-1}	$+$	$+$	$+$
P_i	$+$	0	$-$
P_{i+1}	$-$	$-$	$-$
$W(x)$	1	1	1

K navýšení počtu znaménkových změn tak dochází jenom při přechodu přes kořen polynomu P_0, odku bezprostředně plyne tvrzení věty.

Příklad 1. Zjistíme počet kladných a záporných kořenů polynomu $x^4 - 4x^2 + 8x - 2$. Nejprve sestrojím Sturmovu posloupnost:

```
>> P0=[1 -4 0 8 -2];
>> P1=-polyder(P0)
P1 =
    -4    12     0    -8
```

Protože nás zajímají jen znaménka a jejich zmeny, je možné polynom vydělit nebo vynásobit kladným číslem. To se hodí hlavně při ručním počítání, když se chceme vyhnout složitějším zlomkům. Pak dál pokračujeme v kostrukci Sturmovy posloupnosti: provedeme dělení se zbytkem, odstraníme nulové koeficienty a polynom opět můžeme vydělit kladným číselm:

```
>> P1=P1/4
P1 =
    -1     3     0    -2
>> [Q,R]=deconv(P0,P1)
Q =
    -1     1
R =
     0     0    -3     6     0
>> P2=-R/3;
>> P2(1:2)=[]
P2 =
     1    -2     0
```

Celou činnost opakujeme, dokud nedostaneme konstantní polynom:

```
>> [Q,R]=deconv(P1,P2)
Q =
    -1     1
R =
     0     0     2    -2
>> P3=-R/2;
>> P3(1:2)=[]
P3 =
    -1     1
>> [Q,R]=deconv(P2,P3)
Q =
    -1     1
```

```
R =
     0     0    -1
>> P4=1;
```

Podle předchozího textu je horní hranice pro absolutní hodnotu všech kořenů rovna 9, takže všechny kořeny leží v intervalu $[-9, 9]$. Při ručním počítání je jednodušší určit znaménko limity hodnoty v $\pm\infty$ podle parity nejvyšší mocniny a znaménka jejího koeficientu. Počet znaménkových změn v Matlabu určíme pomocí příkazů:

```
>> x=-9;
>> [polyval(P0,x),polyval(P1,x),polyval(P2,x),...
polyval(P3,x),polyval(P4,x)]
ans =
     9403      970       99       10        1
>> x=0;
>> [polyval(P0,x),polyval(P1,x),polyval(P2,x),...
polyval(P3,x),polyval(P4,x)]
ans =
    -2    -2     0     1     1
>> x=9;
>> [polyval(P0,x),polyval(P1,x),polyval(P2,x),...
polyval(P3,x),polyval(P4,x)]
ans =
     3715     -488       63       -8        1
```

Při ručním počítání stačí určovat znaménka a výsledky můžeme přehledně umístit do tabulky:

x	$P0(x)$	$P1(x)$	$P2(x)$	$P3(x)$	$P4(x)$	$W(x)$
$-\infty$	$+$	$+$	$+$	$+$	$+$	0
0	$-$	$-$	0	$+$	$+$	1
∞	$+$	$-$	$+$	.	$+$	4

Celkem tedy máme 4 kořeny, z toho je jeden záporný a tři kladné.

3. Interpolace

Problém interpolace patří do teorie aproximace. Zpravidla se snažíme aproximovat nějakou funkci, z níž známe jenom hodnoty v dikrétních bodech, někdy je dokonce můžeme mít nepřesné.

Předpokládejme, že jsou dány body x_i, $i = 0, 1, \ldots, n$, $x_i \neq x_k$ pro $i \neq k$ a hodnoty funkce f v těchto bodech: $f(x_i) = f_i$, $i = 0, 1, \ldots, n$. Hledáme polynom P_n stupně nejvýše n takový, že

$$P_n(x_i) = f_i, \qquad i = 0, 1, \ldots, n.$$

Body x_i, $i = 0, 1, \ldots, n$, $x_i \neq x_k$ pro $i \neq k$, budeme nazývat *uzly*, polynom P_n *interpolační polynom*. Platí následující tvrzení, které se dá poměrně snadno dokázat:

Pro $(n+1)$ *daných dvojic čísel*

$$(x_i, f_i), \qquad i = 0, 1, \ldots, n, \quad x_i \neq x_k \text{ pro } i \neq k,$$

existuje nejvýše jeden polynom P_n *stupně menšího nebo rovného* n *takový, že*

$$P_n(x_i) = f_i, \qquad i = 0, 1, \ldots, n.$$

Pro důkaz existence interpolačního polynomu použijeme Lagrangeovu konstrukci, podle níž se interpolační polynom nazývá Lagrangův:

Položme

$$l_i(x) = \frac{(x - x_0) \ldots (x - x_{i-1})(x - x_{i+1}) \ldots (x - x_n)}{(x_i - x_0) \ldots (x_i - x_{i-1})(x_i - x_{i+1}) \ldots (x_i - x_n)} = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}.$$

Je zřejmé, že l_i je polynom stupně n a

$$l_i(x_j) = \begin{cases} 0 & \text{pro } i \neq j \\ 1 & \text{pro } i = j. \end{cases}$$

Definujme nyní Lagrangův interpolační polynom P_n vztahem:

$$P_n(x) = l_0(x) f_0 + l_1(x) f_1 + \ldots + l_n(x) f_n = \sum_{i=0}^{n} l_i(x) f_i.$$

Snadno se ověří, že tento polynom splňuje dané interpolační podmínky a je polynomem stupně nejvýše n.

Polynomy l_i se nazývají Lagrangeovy fundamentální polynomy a tvoří bázi prostoru polynomů stupně nejvýše n.

Podívejme se, jaké je výpočetní složitost Konstrukce Lagrangeova interpolačního polynomu. Při konstrukci jednoho fundamentálního polynomu začínáme polynomem prvního stupně $x - x_0$, který postupně násobíme členy $x - x_j$, stupeň polynomu se postupně zvětšuje a pro jeho násobení potřebujeme řádově tolik operaci, jaký je jeho stupeň. Celkový počet operací pro konstrukci l_i tedy dostaneme jako součet konečné aritmetické řady, což je řádově n^2 operací. Abychom sestrojili všechny fundamentální polynomy potřebujeme tedy řádově n^3 operací.

Při pruční konstrukci interpolačního polynomu ovšem zjistíme, že spoustu operací provádím opakovaně, takže při vhodném postupu by bylo možné konstrukci urychlit. A skutečně, optimální konstrukce Lagrangova interpolačního polynomu je založena na funkci $\omega_{n+1}(x) = (x -$

$x_0) \dots (x - x_n) = \prod_{i=0}^{n}(x - x_i)$. Pro určení funkce ω_{n+1} potřebujem řádově také n^2 oprací, ale tuto funci konstruujeme jen jedenkrát. Čitatel fundamentálního polynomu l_i získáme dělením $\omega_{n+1}(x) : (x - x_i)$, kčemuž se dá využít Hornerovo schema, které je co se týče počtu operací lineární.

Dále se dá odvodit, že jmenovatel fundamentálního polynomu l_i je roven derivaci funkce ω_{n+1} v bodě x_i. Pro derivování polynomu je potřeba také řádově n operací a pro výpočet hodnoty derivace v bodě jakbysmet. Celkově tedy pro jeden fundamentální polynom potřebujeme jen lineární množství operací, pro všechny fundamentální polynomy je to pak řádově n^2 operací.

Matlabovská funkce pro konstrukci Lagrangeova fundamentálního polynomu pak může vypadat následovně:

```
function [lip,lfp] = laintpol2(uzly,ff)
% function [lip,lfp] = laintpol(uzly,ff)
% Lagrangeuv interpolacni polynom
% uzly,ff - radkove vektory
% lip - Lagr. interpol. polynom
% lfp - matice fundamentalnich polynomu

om=poly(uzly);   % funkce omega
omd=polyder(om); % omega'
n=length(uzly)-1;
lip=zeros(1,n+1);
lfp=[];
for ii=0:n
    i1=ii+1;
    xi=uzly(i1);
    citatel=deconv(om,[1,-xi]);
    jmenovatel=polyval(omd,xi);
    li=1/jmenovatel*citatel; % fundamentalni polynom
    lfp=[lfp;li];
end % konec cyklu
lip=ff*lfp;
end % konec funkce
```

Literatura

[Horová and Zelinka, 2008] Horová, I. and Zelinka, J. (2008). *Numerické metody*. Masarykova univerzita, Brno, 2. rozšířené edition.