NA09 Interpolacijski polinomi

Ivan Slapničar

5. studenog 2018.

1 Interpolacijski polinomi

Neka je zadana $n+1$ točka

$$T_i=(x_i,y_i),\quad i=0,1,\ldots,n,\quad x_i\neq x_j.$$

1.1 Standardna baza

Kroz zadane točke prolazi *interpolacijski polinom* $p_n(x)$. Koeficijenti polinoma zadovoljavaju sustav linearnih jednadžbi $p_n(x_i)=y_i$, $i=0,\ldots,n$, odnosno

$$\begin{bmatrix}1 & x_0 & x_0^2 & x_0^3 & \cdots & x_0^n\\ 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^n\\ \vdots & & & & \vdots & \\ 1 & x_n & x_n^2 & x_n^3 & \cdots & x_n^n\end{bmatrix}\begin{bmatrix}a_0\\ a_1\\ \vdots\\ a_n\end{bmatrix}=\begin{bmatrix}y_0\\ y_1\\ \vdots\\ y_n\end{bmatrix}$$

Matrica sustava A se zove *Vandermonde-ova matrica*. Njena determinanta dana je formulom

$$\det(A)=\prod_{0\leq j<i\leq n}(x_i-x_j).$$

Kako su sve apscise različite ($x_i\neq x_j$ za $i\neq j$), vrijedi $\det(A)\neq 0$ pa je matrica A regularna i zadani sustav ima jedinstveno rješenje - dakle,

> interpolacijski polinom je *jedinstven*.

In [1]:
```
using Polynomials
using Gadfly
```

In [2]:
```
# Generirajmo slučajne točke
srand(123)
n=6
x=rand(n)
y=rand(n)
```

```
a=minimum(x)
b=maximum(x)
```

Out[2]: 0.940515000715187

In [3]:
```
# Ova datoteka omogućuje manipulaciju s Vandermondeovim matricama
include("Vandermonde.jl")
```

Out[3]: full (generic function with 1 method)

In [4]:
```
A=Vandermonde(x)
```

Out[4]:
```
6×6 Vandermonde{Float64}:
 1.0  0.768448  0.590512   0.453777   0.348704    0.267961
 1.0  0.940515  0.884568   0.83195    0.782461    0.735917
 1.0  0.673959  0.45422    0.306126   0.206316    0.139049
 1.0  0.395453  0.156383   0.0618422  0.0244557   0.00967108
 1.0  0.313244  0.0981218  0.0307361  0.00962788  0.00301588
 1.0  0.662555  0.438979   0.290848   0.192702    0.127676
```

In [5]:
```
c=A\y
```

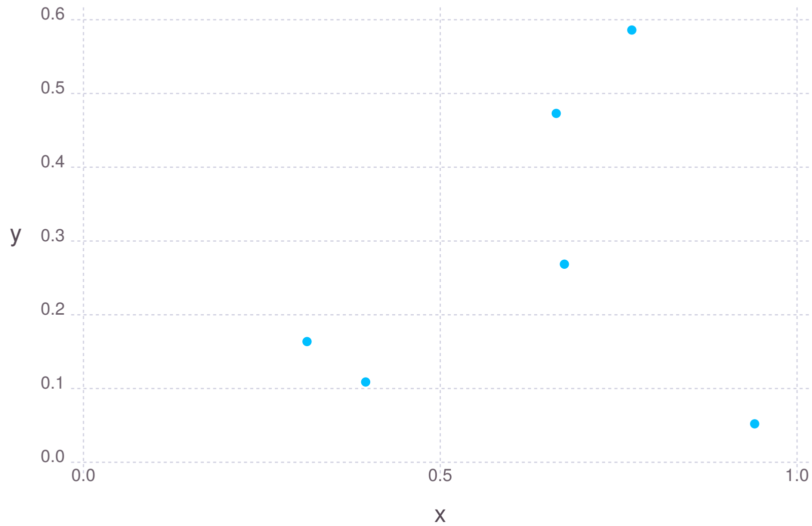
Out[5]:
```
6-element Array{Float64,1}:
    461.086
  -4416.86
  16180.8
 -28352.8
  23866.6
  -7754.35
```

In [6]:
```
p=Poly(c)
```

Out[6]:
```
Poly(461.0864919227929 - 4416.86065664229*x + 16180.757514746274*x^2
 - 28352.751446521183*x^3 + 23866.64457655297*x^4 - 7754.352137939532*x^5)
```

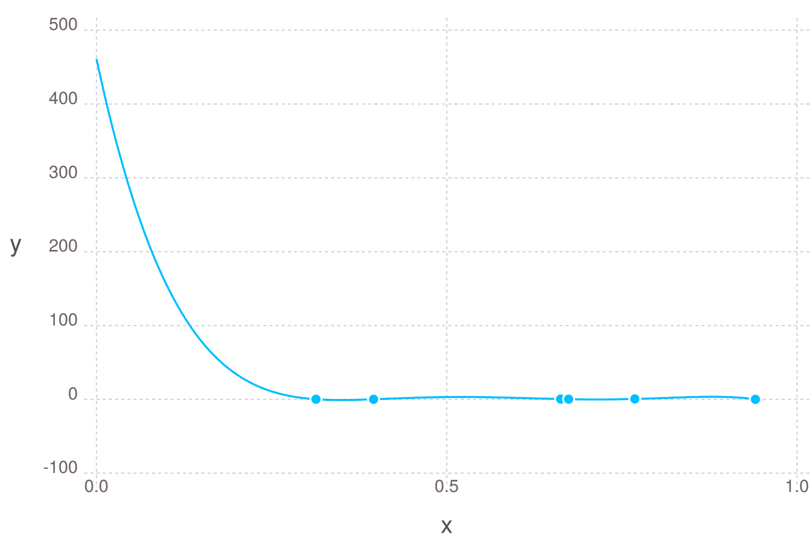
In [7]:
```
plot(x=x,y=y)
```

Out[7]:



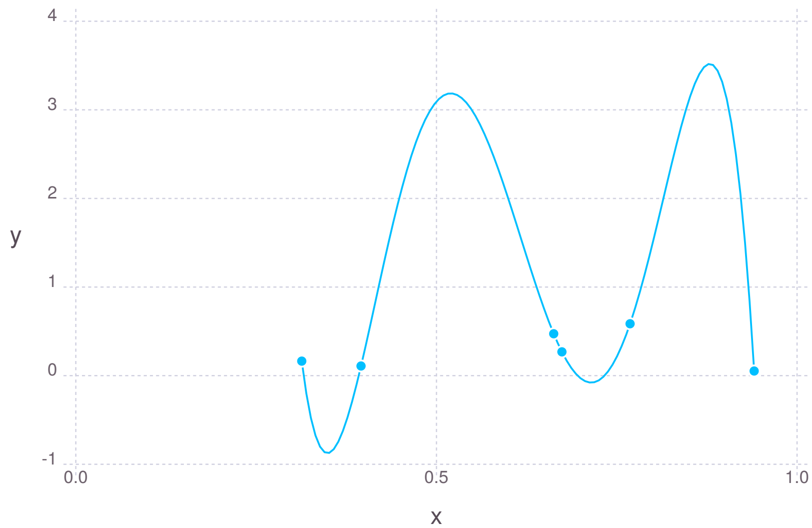

```
In [8]: plot(layer(x=x,y=y,Geom.point),layer(x->p(x),0,0.95,Geom.line))
```

```
Out[8]:
```



```
In [9]: xx=linspace(a,b,100)
        pS=polyval(p,xx)
        plot(layer(x=x,y=y,Geom.point),layer(x=xx,y=pS,Geom.line))
```

Out[9]:




```
In [10]: cond(A)
```

Out[10]: 861893.3298937214

Za rješavanje zadanog sustava standardnim putem potrebno je $O(n^3)$ računskih operacija, no postoje metode kojima se Vandermondeovi sustavi mogu riješiti s $O(n^2)$ operacija.

Za izvrednjavanje polinoma u nekoj točki potrebno je $2n$ operacija (Hornerova shema).

Vandermondeova matrice uglavnom imaju veliku kondiciju pa ovaj način računanja koeficijenata polinoma može biti nestabilan. Stoga se koriste i druge metode za računanje i izvredjavanje interpolacijskih polinoma.

1.2 Lagrangeov interpolacijski polinom

Definirajmo $n+1$ polinom stupnja n:

$$L_j(x) = \prod_{\substack{i=0\\ i\neq j}}^{n} \frac{x-x_i}{x_j-x_i}.$$

Vrijedi

$$L_j(x_i) = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$$

pa je

$$p_n(x) = y_0\, L_0(x) + y_1\, L_1(x) + \cdots + y_n\, L_n(x).$$

Za računanje nazivnika polinoma prvi put je potrebno $O(n^2)$ operacija, ali se potom vrijednost $p_n(x)$ računa s $O(n)$ operacija (*objasnite kako!*).

Navodimo implementaciju algoritma koja nije optimalno brza.

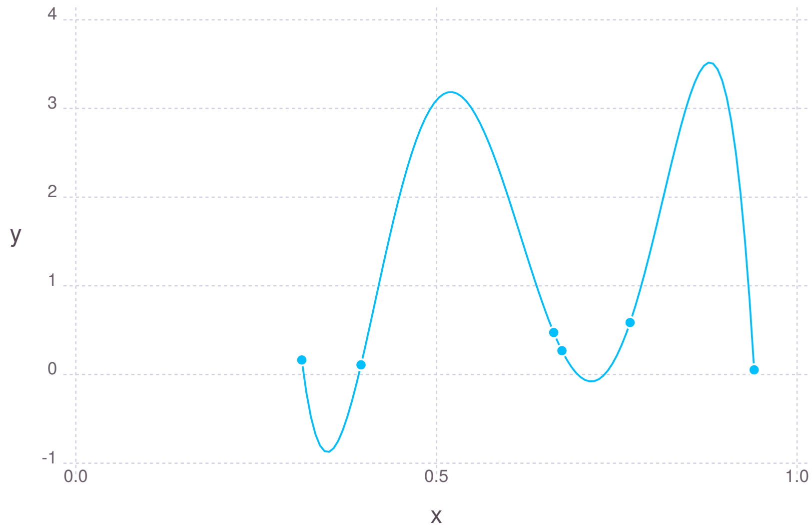
```
In [11]: L(t)=sum(y.*[prod(t-x[[1:j-1;j+1:end]])/prod(x[j]-x[[1:j-1;j+1:end]])
                 for j=1:n])
```

```
Out[11]: L (generic function with 1 method)
```

```
In [12]: pL=Array{Float64}(length(xx))
         for i=1:length(xx)
             pL[i]=L(xx[i])
         end
```

```
In [13]: plot(layer(x=x,y=y,Geom.point),layer(x=xx,y=pS,Geom.line))
```

```
Out[13]:
```




```
In [14]: norm(pS-pL,Inf)
```

```
Out[14]: 3.191802377955355e-11
```

```
In [15]: norm(abs.((pS-pL)./pL),Inf)
```

```
Out[15]: 1.762269768405905e-10
```

1.3 Newtonov interpolacijski polinom

Kod ovog polinoma koristi se baza

$$1, x-x_0, (x-x_0)(x-x_1), (x-x_0)(x-x_1)(x-x_2), \ldots, (x-x_0)(x-x_1)\cdots(x-x_{n-1})$$

pa je interpolacijski polinom dan s

$$p_n(x) = c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + \cdots + c_n(x-x_0)(x-x_1)\cdots(x-x_{n-1}).$$

Koeficijenti interpolacijskog polinoma su rješenje *trokutastog* sustava jednadžbi $Lc = y$,

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & x_1-x_0 & 0 & 0 & \cdots & 0 \\ 1 & x_2-x_0 & (x_2-x_0)(x_2-x_1) & 0 & \cdots & 0 \\ \vdots & & & & \vdots & \\ 1 & x_n-x_0 & (x_n-x_0)(x_n-x_1) & (x_n-x_0)(x_n-x_1)(x_n-x_2) & \cdots & (x_n-x_0)\cdots(x_n-x_{n-1}) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

Za formiranje donje trokutaste matrice L potrebno je $O(n^2)$ operacija. Za računanje koeficijenata $c_0, \ldots, c_n$ potrebno je $O(n^2)$ operacija (rješavanje donje trokutastog sustava) i to rješenje je *stabilno*.

Za računanje $p_n(x)$ koristi se postupak koji je vrlo sličan Hornerovoj shemi.

```
In [16]: # Računanje koeficijenata c
         function mynewton(x,y)
             n=length(x)
             L=zeros(n,n)
             L[:,1]=ones(n)
             for i=2:n
                 for j=2:i
                     L[i,j]=prod([x[i]-x[k] for k=1:j-1])
                 end
             end
             c=L\collect(y)
         end
```

```
Out[16]: mynewton (generic function with 1 method)
```

```
In [17]: c=mynewton(x,y)
```

```
Out[17]: 6-element Array{Float64,1}:
             0.586022
            -3.10279
           -24.2415
           -58.1748
          -106.853
         -7754.35
```

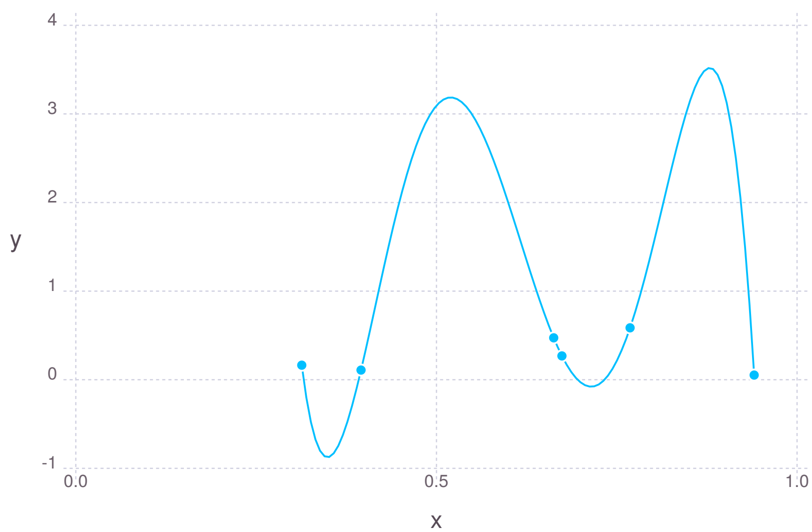
```
In [18]: # Računanje vrijednosti Newtonovog polinoma zadanog s točkama x i
         # koeficijentima c u točki t
         function evalnewton(c,x,t::Number)
             p=c[end]
             for i=length(c)-1:-1:1
                 p=p*(t-x[i])+c[i]
             end
             p
         end
```

```
Out[18]: evalnewton (generic function with 1 method)
```

```
In [19]: pN=Array{Float64}(length(xx))
         for i=1:length(xx)
             pN[i]=evalnewton(c,x,xx[i])
         end
```

```
In [20]: plot(layer(x=x,y=y,Geom.point),layer(x=xx,y=pS,Geom.line))
```

```
Out[20]:
```




```
In [21]: norm(abs.((pS-pN)./pN),Inf)
```

```
Out[21]: 1.7621023159383342e-10
```

```
In [22]: norm(abs.((pL-pN)./pN),Inf)
```

Out[22]: 1.6745246754141728e-14

Vidimo da su pN i pL bliže jedan drugome nego pS pa zaključujemo da su zaista točniji.