Numeričke metode

Aleksandar Maksimović
IRB

Interpolacija, Ekstrapolacija

Znamo vrijednosti funkcije f(x) na skupu točaka $x_1, x_2, x_3, \ldots, x_N$ ali nemamo analitički izraz za f(x) koji bi nam omogućio da izračunamo vrijednost funkcije u bilo kojoj točki.

- Provući glatku krivulju kroz zadane točke
- Ako je željeni *x* između najmanjeg i najvećeg ➔ *Interpolacija*
- *Ako je x izvan toga područja (riskantno)* ➔ *Ekstrapolacija*

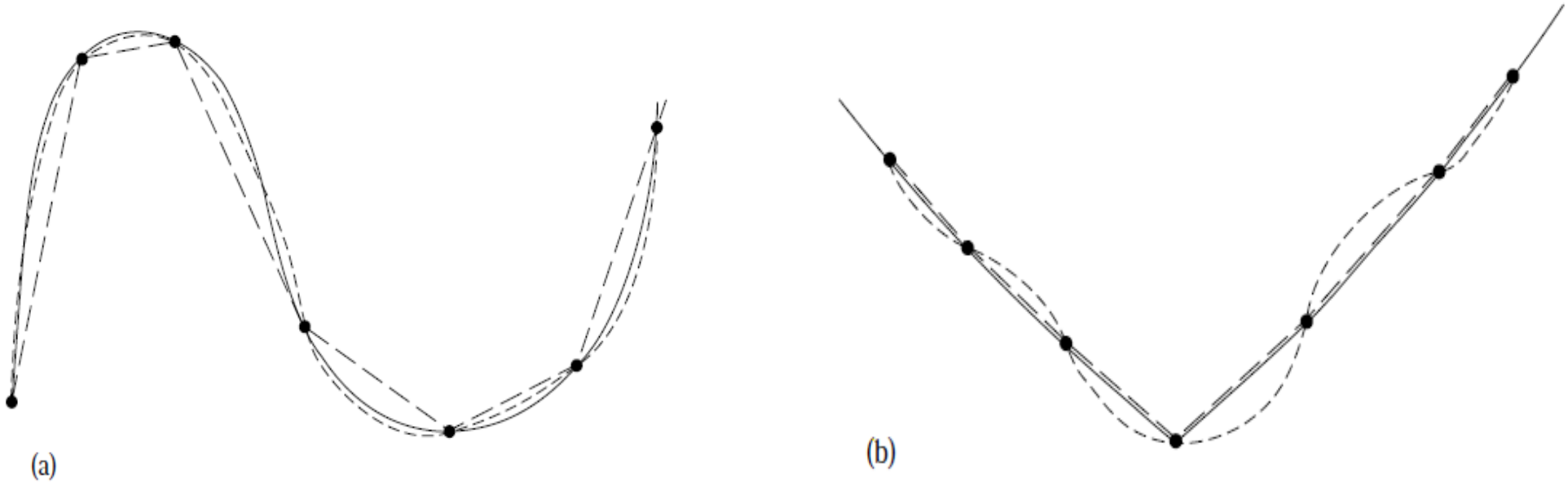
Koristimo slijedeće funkcije:

a) polinome

b) racionalne funkcije (kvocijenti polinoma)

c) trigonometrijske funkcije

Interpolacija



a) Glatka funkcija je bolje interpolirana s polinomom većeg reda

b) funkcija s oštrim kornerom je bolje aproksimirana s polinomom manjeg reda

Eksponencijalne i racionalne funkcije mogu biti loše aproksimirane polinomima većeg stupnja.

Interpolacija polinomom

Interpolacijski polinom

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \sum_{j=0}^{n} a_j x^j$$

nepoznati su koeficijenti, određujemo ih poznatih vrijednosti y

$$p(x_i) = y_i,\ i = 0, \ldots, n,$$

$$\begin{aligned}
a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_{n-1} x_0^{n-1} + a_n x_0^n &= y_0 \\
a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_{n-1} x_1^{n-1} + a_n x_1^n &= y_1 \\
\vdots \quad \vdots \quad \vdots \quad \ldots \quad \vdots \quad \vdots \quad \vdots & \\
a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_{n-1} x_i^{n-1} + a_n x_i^n &= y_i \\
a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_{n-1} x_n^{n-1} + a_n x_n^n &= y_n.
\end{aligned}$$

Interpolacija

$$Va = y$$

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & \cdots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^{n-1} & x_1^n \\ & & \cdots & & \cdots & & \\ 1 & x_i & x_i^2 & x_i^3 & \cdots & x_i^{n-1} & x_i^n \\ & & \cdots & & \cdots & & \\ 1 & x_n & x_n^2 & x_n^3 & \cdots & x_n^{n-1} & x_n^n \end{bmatrix}}_{V} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}}_{a} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}}_{y}.$$

$$D_n = \begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^n \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{vmatrix}.$$

Vandermonde determinanta

Vandermonde sustav je "loše" uvjetovan (ill-conditioned), pogreške zaokruživanja mogu biti velike.

Uvjetovanost sustava

broj uvjetovanosti matrice:

$$\mathrm{cond}(A) = \|A\| \cdot \|A^{-1}\|$$

- singularna matrica $\mathrm{cond}(A) = \infty$
- Linearni sustav je slabo uvjetovan (ill-conditioned) ako je $\mathrm{cond}(A)$ velik.
- Umjesto A^{-1} koeficijenti polinoma se traže alternativnim načinom
  - poglavlje 2.8 u Numerical Recipes

Interpolaciona krivulja prolazi točno kroz zadane točke.
Ako postoji pogreška kod točaka, izračunati koeficijenti su nepouzdani.
RAČUNANJE KOEFICIJENATA TREBA IZBJEGAVATI.

Podprogrami

SUBROUTINE polcoe(x,y,n,cof)

PARAMETER (NMAX=15) Largest anticipated value of n.

Given arrays x(1:n) and y(1:n) containing a tabulated function $y_i = f(x_i)$, this routine returns an array of coecients cof(1:n), such that $y_i = \sum_j cof_j x_i^{j-1}$

void polcoe(float x[], float y[], int n, float cof[])

Given arrays x[0..n] and y[0..n], returns cof[0..n]

SUBROUTINE polcof(xa,ya,n,cof)

C USES polint

Given arrays xa(1:n) and ya(1:n) of length n containing a tabulated function $ya_i = f(xa_i)$, this routine returns an array of coecients cof(1:n), also of length n, such that $ya_i = \sum_j cof_j xa_i^{j-1}$.

void polcof(float xa[], float ya[], int n, float cof[])

Given arrays xa[0..n] and ya[0..n], returns an array of coecients cof[0..n]

NR primjeri

NR primjeri:

C;

```
gcc -o polcoeC xpolcoe.c polcoe.c  nrutils/nrutil.c libnr.c  -lm -I nrutils/
gcc -o polcofC xpolcof.c polcof.c polint.c  nrutils/nrutil.c libnr.c  -lm -I nrutils/
```

F77:

```
f77 -o xpolcoeF  xpolcoe.for polcoe.for
f77 -o xpolcofF xpolcof.for polcof.for polint.for
```

Primjeri pokazuju interpolaciju sin(x) na intervalu od 0 do Pi, exp(x) na intervalu od 0 do 1

Lagrange polinom

Interpolirajući polinom stupnja *N-1* kroz *N* točaka ima Lagrangeov oblik:

$$P(x) = \frac{(x-x_2)(x-x_3)...(x-x_N)}{(x_1-x_2)(x_1-x_3)...(x_1-x_N)}y_1 + \frac{(x-x_1)(x-x_3)...(x-x_N)}{(x_2-x_1)(x_2-x_3)...(x_2-x_N)}y_2$$

$$+ \cdots + \frac{(x-x_1)(x-x_2)...(x-x_{N-1})}{(x_N-x_1)(x_N-x_2)...(x_N-x_{N-1})}y_N$$

Upotrebljavanje Lagrangeove formule je "teško" programirati

Neville-ov algoritam (sličan njemu je i Aitkenov algoritam)

$$\begin{array}{lll} x_1: & y_1 = P_1 & \\ & & P_{12} \\ x_2: & y_2 = P_2 & P_{123} \\ & & P_{23} \\ x_3: & y_3 = P_3 & P_{234} \\ & & P_{34} \\ x_4: & y_4 = P_4 & \end{array} \qquad P_{1234}$$

- Rekurzivan način punjenja (stupac po stupac)
- Bazira se na relaciji “kćerke” P i dva “roditelja”.

Joseph-Louis Lagrange (1736-1813)

Italian-French mathematician associated with many classic mathematics and physics – Lagrange multipliers in minimization of a function, Lagrange's interpolation formula, Lagrange's theorem in group theory and number theory, and the Lagrangian ($L=T-V$) in mechanics and Lagrange equations.

Određivanje polinoma

- Konstruiraju se vrijednosti u zadanim točkama (x_i,y_i), koje označimo P_1 ,P_2 , ...,P_N
- Tražimo polinom prvog stupnja koji prolazi kroz prve dvije točke
- Za vrijednosti P_1 i P_2 u $x=x_1$ i x_2, imamo $P_{12}(x) = \lambda(x)\,P_1+[1-\lambda(x)]\,P_2$

  Zahtjevamo $P_{12}(x_1) = P_1$ and $P_{12}(x_2) = P_2$, to je ispunjeno ako vrijedi

  $$\lambda(x_1) = 1,\ \lambda(x_2) = 0 \ \text{ili}\ \ \lambda(x) = (x-x_2)/(x_1-x_2)$$

- Slijedeći stupanj konstruiramo iz dobivene dvije vrijednosti

  $$P_{123}(x) = \lambda(x)\,P_{12}(x) + [1-\lambda(x)]\,P_{23}(x)$$

  $P_{123}(x_2) = P_2$ ispunjeno za svaki $\lambda(x)$.

Interpolacija Lagrange

Zahtjevamo $P_{123}(x_1)=P_{12}(x_1)=P_1$ i $P_{123}(x_3)=P_{23}(x_3)=P_3$,

mora vrijediti $\lambda(x_1)=1$, $\lambda(x_3)=0$, odnosno $\lambda(x)=(x-x_3)/(x_1-x_3)$

Rekurzija:

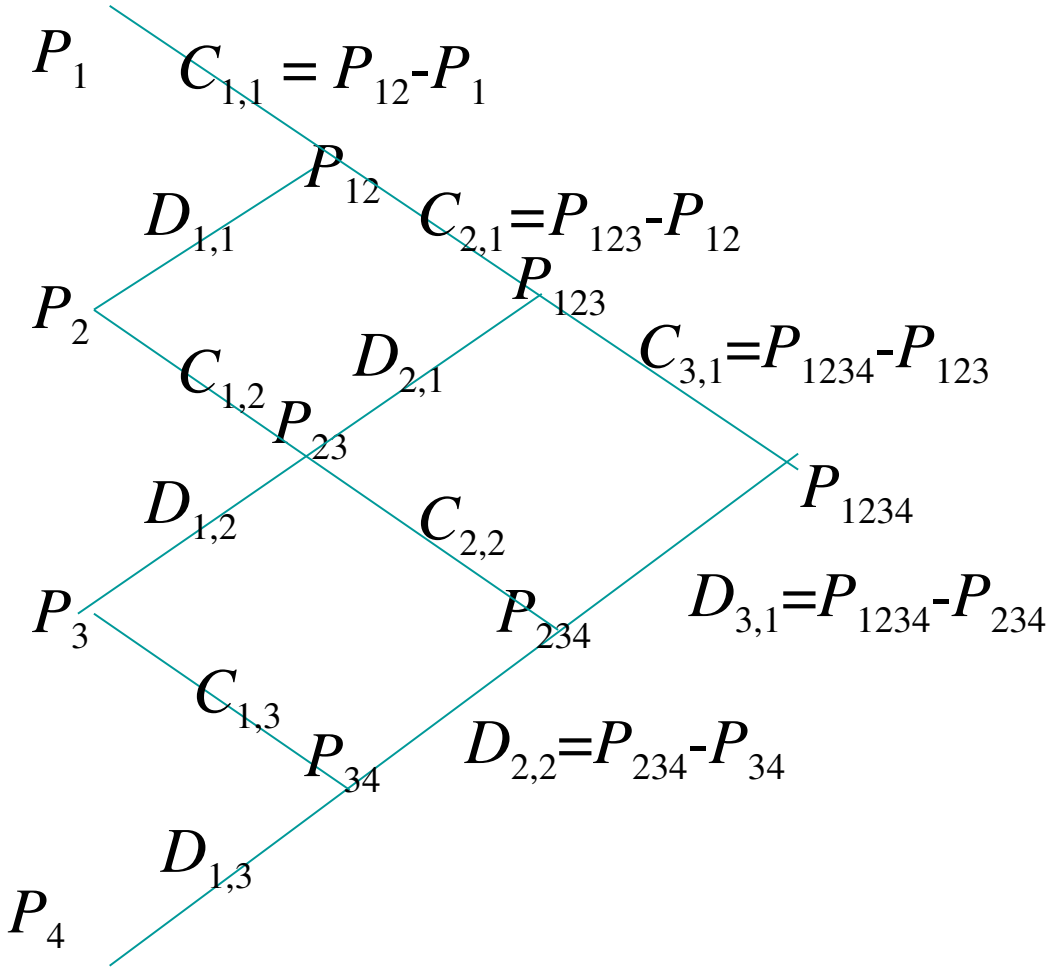
- Dvije interpolirane vrijednosti s m-točaka P, konstruirana iz točaka $i,i+1,i+2,\ldots,i+m-1$, i $i+1,i+2,\ldots,i+m$, određuju slijedeći nivo $m+1$ točaka interpolacije od i do $i+m$:

$$P_{i(i+1)(i+2)\ldots(i+m)}(x) = \lambda(x)\, P_{i(i+1)(i+2)\ldots(i+m-1)}(x) + [1-\lambda(x)]\, P_{(i+1)(i+2),\ldots(i+m)}(x)$$

$$\lambda(x) = (x-x_{i+m})/(x_i-x_{i+m}) \quad \text{ili raspisano}$$

$$P_{i(i+1)\ldots(i+m)} = \frac{(x-x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i-x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i-x_{i+m}}$$

Lagrange polinom



Aleksandar Maksimović,
Institute Ruđer Bošković, Bijenička c. 54

Interpolacija Lagrange

Prati se razlika između kćeri i roditelja, tj. definiraju se veličine:

$$C_{m,i} \equiv P_{i\ldots(i+m)} - P_{i\ldots(i+m-1)}$$

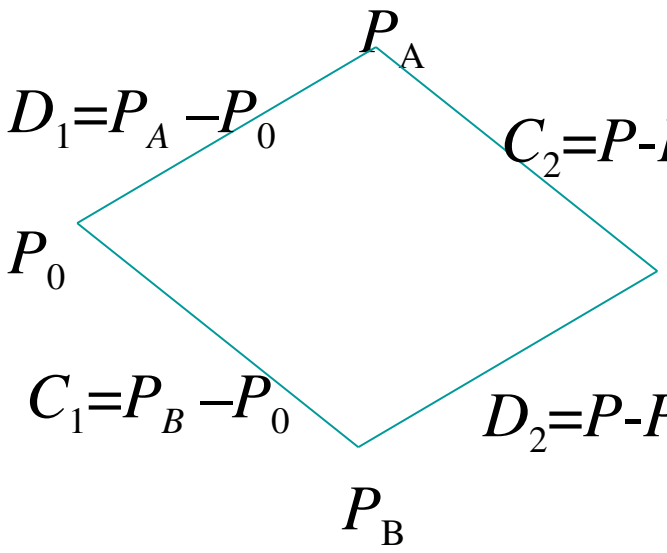
$$D_{m,i} \equiv P_{i\ldots(i+m)} - P_{(i+1)\ldots(i+m)}.$$

Na isti način kao kod popunjavanja tablice interpolacije možemo dobiti relacije

$$D_{m+1,i} = \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}$$

$$C_{m+1,i} = \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}$$

Izvod C,D koeficijenta



$$P = \lambda\ P_A + (1-\lambda)P_B$$

$$C_2 = P-P_A = P-P_0 +P_0 -P_A = P-P_0 -(P_A -P_0)$$

$$C_2 = \lambda\ P_A + (1-\lambda)P_B -P_0 -(P_A -P_0) =(1-\lambda)(C_1 -D_1)$$

Podprogrami: Lagrange interpolacija

SUBROUTINE polint(xa,ya,n,x,y,dy)

PARAMETER (NMAX=10) Largest anticipated value of n.

Given arrays xa and ya, each of length n, and given a value x, this routine returns a value y, and an error estimate dy. If P(x) is the polynomial of degree N − 1 such that $P(\mathtt{xa}_i) = \mathtt{ya}_i$; i = 1,...,n, then the returned value y = P(x).

void polint(float xa[], float ya[], int n, float x, float *y, float *dy)

Given arrays xa[1..n] and ya[1..n], and given a value x, this routine returns a value y, and an error estimate dy.

Podprogrami

NR primjer:
gcc -o polintC xpolint.c polint.c nrutils/nrutil.c -I nrutils/ -lm
interpolira sin(x) i exp(x) od Pi/n do Pi, tj od 1/n do 1.
f77 -o xpolintF xpolint.for polint.for

Interpolacija tablice:
gcc -o tpolintC tpolint.c polint.c nrutils/nrutil.c libnr.c -lm -I nrutils/
otvori Log10.dat, pozovi polint i spremi u Interp.dat, funkcija log(x) iz intervala 0.1 do 10

NAPOMENA: U polint.for je izmjenjen NMAX od 10 na 50, tj. prije je bila dopuštena interpolacija od samo 10 točaka. polint.c nema takvih problema zbog dinamičkog alociranja vektora.

Interpolacija racionalnim funkcijama

Neke funkcije se bolje aproksimiraju s racionalnim funkcijama, npr. kvocijenti polinoma.

Racionalne funkcije opisuju veći broj krivulja, manje osciliraju.

Racionalna funkcija koja prolazi kroz *m+1* točku je:

$$R_{i(i+1)\ldots(i+m)} = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{p_0 + p_1 x + \cdots + p_\mu x^\mu}{q_0 + q_1 x + \cdots + q_\nu x^\nu}$$

P polinom stupnja μ

Q polinom stupnja ν

Imamo ν+μ+1 nepoznanica (*p*,*q*, i q_0 je proizvoljan)

mora vrijediti: $m + 1 = \mu + \nu + 1$

- U specifikaciji racionalne funkcije treba dati željeni stupanj brojnika i nazivnika
- Racionalne funkcije mogu modelirati funkcije s polom tj s nulama polinoma u brojniku kod izraza s lijeve strane
- Bulirsch i Stoer su razvili algoritam sličan Nevillev-om

Interpolacija racionalnim funkcijama

Konstruira se racionalna funkcija koja prolazi kroz zadani skup točaka
Konstruira se tablica racionalnih funkcija stupac po stupac kao kod Nevillovog algoritma. Rekurzija za racionalnu funkciju je:

$$R_{i(i+1)\ldots(i+m)} = R_{(i+1)\ldots(i+m)} + \frac{R_{(i+1)\ldots(i+m)} - R_{i\ldots(i+m-1)}}{\left(\frac{x-x_i}{x-x_{i+m}}\right)\left(1 - \frac{R_{(i+1)\ldots(i+m)} - R_{i\ldots(i+m-1)}}{R_{(i+1)\ldots(i+m)} - R_{(i+1)\ldots(i+m-1)}}\right) - 1}$$

Ponovo se rekurzija može napisati pomoću razlika "roditelja" i "kćeri", tj. preko koeficijenata C i D

$$C_{m,i} \equiv R_{i\ldots(i+m)} - R_{i\ldots(i+m-1)}$$

$$D_{m,i} \equiv R_{i\ldots(i+m)} - R_{(i+1)\ldots(i+m)}$$

podprogrami

SUBROUTINE ratint(xa,ya,n,x,y,dy)

PARAMETER (NMAX=10,TINY=1.e-25) Largest expected value of n, and a small number.

Given arrays xa and ya, each of length n, and given a value of x, this routine returns a value of y and an accuracy estimate dy. The value returned is that of the diagonal rational function, evaluated at x, which passes through the n points $(\mathtt{xa}_i, \mathtt{ya}_i)$, $i = 1...\mathtt{n}$.

void ratint(float xa[], float ya[], int n, float x, float *y, float *dy)

Given arrays xa[1..n] and ya[1..n], and given a value of x, this routine returns a value of y and an accuracy estimate dy.

NR primjer: interpolira x*exp(-x)/(SQR(x-1.0)+eps*eps); u intervalu 0.2 do 2

gcc -o ratintC ratint.c xratint.c nrutils/nrutil.c -I nrutils/ -lm

f77 -o ratintF ratint.for xratint.for

Kubična splajn interpolacija

➢U situacijama gdje je važna kontinuiranost derivacija funkcija moramo koristiti "spline" funkcije !

Promatramo interval određen s dvije susjedne točke, linearna interpolacija daje

$$y = Ay_j + By_{j+1}$$

$$A \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j} \qquad B \equiv 1 - A = \frac{x - x_j}{x_{j+1} - x_j}$$

Specijalan slučaj interpolacije Lagrange

Splajn funkcija je suma kubičnih polinoma oblika

$$S(x) = \begin{cases} s_1(x) & \text{if} \quad x_1 \le x < x_2 \\ s_2(x) & \text{if} \quad x_2 \le x < x_3 \\ \vdots & \\ s_{n-1}(x) & \text{if} \quad x_{n-1} \le x < x_n \end{cases}$$

$$s_i(x) = a_i + b_i(x\text{-}x_i) + c_i(x\text{-}x_i)^2 + d_i(x\text{-}x_i)^3$$

Kubična splajn interpolacija

- Za N točaka (x_i,y_i), i=1,2,…,N, za svaki interval i do i+1, napravi kubični polinom tako da vrijedi $s_i(x_i)=y_i$ and $s_i(x_{i+1})=y_{i+1}$.
- 1. i 2. derivacija moraju biti kontinuirane u intervalu, tj., $s_i^{(n)}(x_{i+1}) = s^{(n)}_{i+1}(x_{i+1})$, n = 1 and 2.
- Rubni uvjet $s''(x_{1 \text{ ili } N})=0$, or $s''(x_{1 \text{ ili } N}) =$ konstanta.

Kubični polinom koji zadovoljava uvjete ima oblik:

$$y = Ay_j + By_{j+1} + Cy''_j + Dy''_{j+1}$$

$$C \equiv \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2 \qquad D \equiv \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2$$

Polinom ispunjava cilj kubične splajn interpolacije: 1.) glatka interpolacija u prvoj derivaciji, 2.) kontinuirana u drugoj derivaciji, na intervalu i na granici.

Kubična splajn interpolacija

Prva derivacija je

$$\frac{dy}{dx} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{3A^2 - 1}{6}(x_{j+1} - x_j)y''_j + \frac{3B^2 - 1}{6}(x_{j+1} - x_j)y''_{j+1}$$

druga derivacija

$$\frac{d^2y}{dx^2} = Ay''_j + By''_{j+1}$$

Druga derivacija se dobiva iz jednakosti prve derivacije u točki $x=x_j$ iz dva susjedna intervala, nakon uređivanja:

$$\frac{x_j - x_{j-1}}{6}y''_{j-1} + \frac{x_{j+1} - x_{j-1}}{3}y''_j + \frac{x_{j+1} - x_j}{6}y''_{j+1} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}$$

Imamo N-2 linearne jednadžbe i N nepoznanica (druge derivacije).

1. **Prirodni kubični splajn**, druge derivacije na rubu su jednake nula
2. Vrijednosti druge derivacije na rubu se računaju iz zadanih prvih derivacija

Podprogrami

SUBROUTINE spline(x,y,n,yp1,ypn,y2)

Given arrays x(1:n) and y(1:n) containing a tabulated function, i.e., yi = f(xi), with $x_1 < x_2 < \ldots < x_N$, and given values yp1 and ypn for the first derivative of the interpolating function at points 1 and n, respectively, this routine returns an array y2(1:n) of length n which contains the second derivatives of the interpolating function at the tabulated points xi. If yp1 and/or ypn are equal to 1×10^{30} or larger, the routine is signaled to set the corresponding boundary condition for a natural spline, with zero second derivative on that boundary. Parameter: NMAX is the largest anticipated value of n.

void spline(float x[], float y[], int n, float yp1, float ypn, float y2[])

SUBROUTINE splint(xa,ya,y2a,n,x,y)

Given the arrays xa(1:n) and ya(1:n) of length n, which tabulate a function (with the xai 's in order), and given the array y2a(1:n), which is the output from spline above, and given a value of x, this routine returns a cubic-spline interpolated value y.

void splint(float xa[], float ya[], float y2a[], int n, float x, float *y)

primjeri

NR primjer:

gcc -o splintC spline.c splint.c xsplint.c nrutils/nrutil.c -I nrutils/ -lm

isto kao kod xpolint.c, samo su vektori duljine 10, i potrebne su derivacije u 1 i zadnjoj tocki.

```
spline(xa,ya,NP,yp1,ypn,y2);
splint(xa,ya,y2,NP,x,&y);
```

F77:

f77 -o splintF xsplint.for splint.for spline.for

C:

gcc -g -o tsplint tsplint.c splint.c spline.c polint.c nrutils/nrutil.c libnr.c -lm -I nrutils/

open Log10.dat, call polint and save in Interpsp.dat, funkciju log(x) iz 0.1 i 10

F77:

f77 -o tsplintF tsplint.for splint.for spline.for lib1.f

Zadaci za praktikum

1. Izvršite programe xpolcoe, xpolcof koji određuju koeficijente polinoma
2. Izvršite programe xpolint i tpolint koji koriste Lagrange interpolaciju
3. Izvršite program xratint, interpolaciju racionalnim funkcijama
4. Izvršite programe xsplint, tsplint za kubičnu splajn interpolaciju
5. Notebook PlotTable.nb za program Mathematica koristite kako bi usporedili krivulje dobivene splajn i Lagrange interpolacijom za funkciju log(x). Log10.dat sadrži tablicu s 40 točaka ove funkcije.
6. Notebook PolCoeff.nb ilustrira polinom interpolacije 4 stupnja. Provjerite da li povećanje stupnja polinoma daje bolju ekstrapolaciju.
7. Napravite Lagrange i splajn interpolaciju na Runge.dat podacima. Nacrtajte krivulje. (1 ili 3 .nb u attachment) Aleksandar.Maksimovic@irb.hr

Literatura

- Online literatura:
  - Numerička matematika-osnovni udžbenik, PMF, projekt mzt.
  - Numerical Recipes in C
  - Numerical Recipes in Fortran
- L. F. Shampine, R. C. Allen, Jr., S. Pruess: FUNDAMENTALS OF NUMERICAL COMPUTING, John Wiley & Sons, Inc. (1997)
- George Em Karniadakis and Robert M. Kirby II: Parallel Scientific Computing in C++ and MPI, Cambridge University Press.

/home/maks/numerickemetode/v5/PlotTable.nb

File Edit Cell Format Input Kernel Find Window Help

100% Input << Go Back

```
Directory[]
```

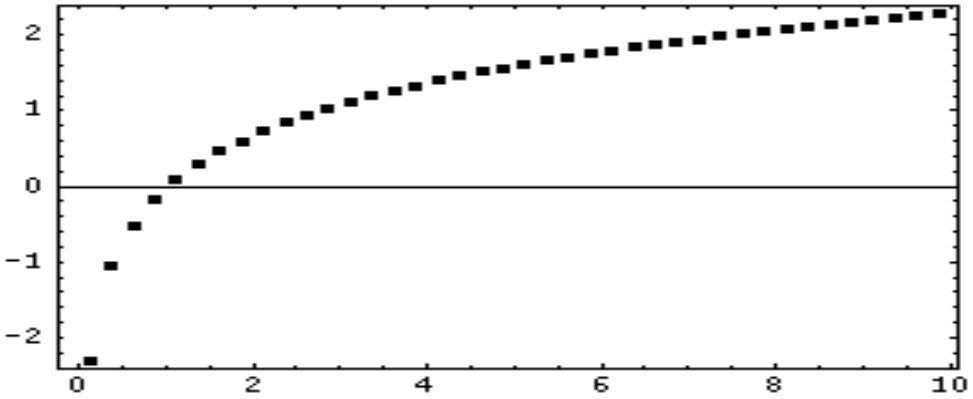
```
/home/maks
```

```
SetDirectory["/home/maks/numerickemetode/v5"]
```

```
/home/maks/numerickemetode/v5
```

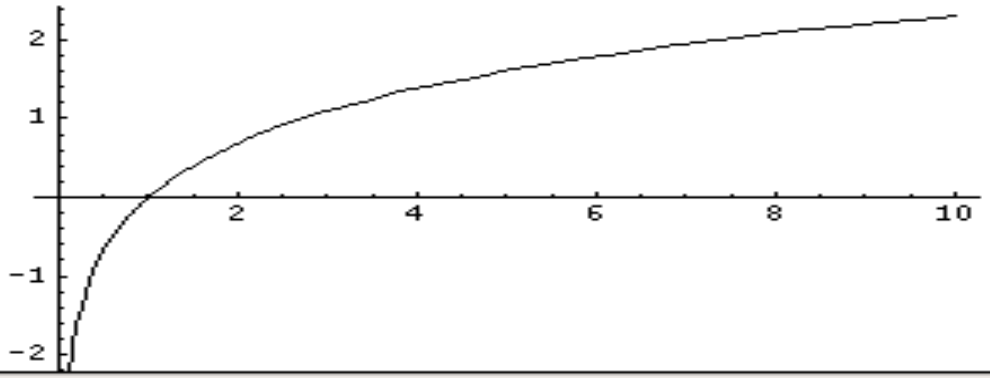
```
xyinterp = Import["InterpspF.dat", "Table"];
```

```
g1 = lPlot[xyinterp]
```



- Graphics -

```
g2 = Plot[Log[x], {x, 0.1, 10}]
```

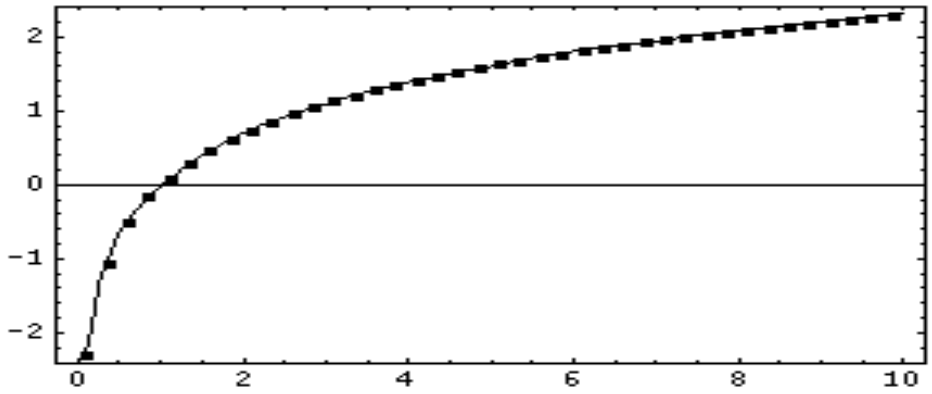


/home/maks/numerickemetode/v5/PlotTable.nb

File Edit Cell Format Input Kernel Find Window Help

100% Input << Go Back

```
Show[g1, g2]
```



- Graphics -

```
lPlot[matr_] :=
  Module[{g1}, g1 = ListPlot[matr, DisplayFunction → Identity, Frame -> True];
   Show[g1, Prolog → AbsolutePointSize[5], DisplayFunction -> $DisplayFunction]];
```

$$f(x) = \frac{1}{1+25x^2}\,, \quad x \in [-1,1]\,. \tag{3.6}$$

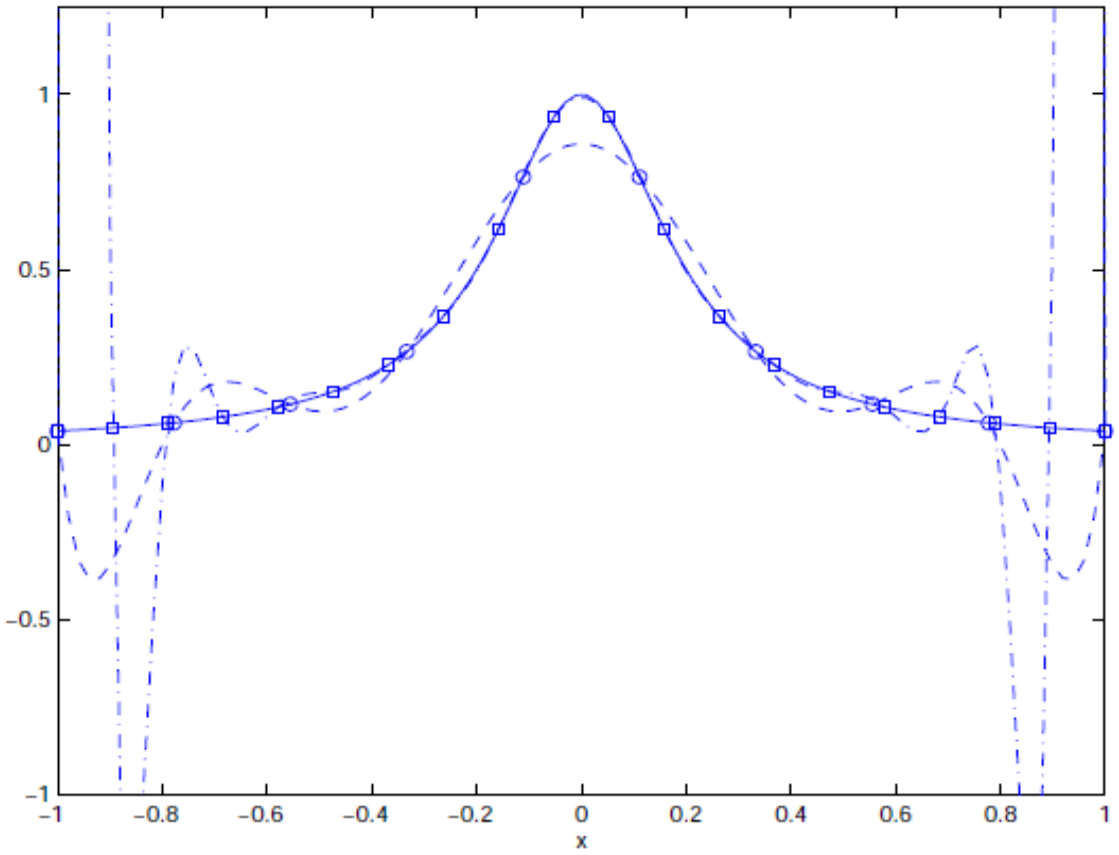


Figure 3.3: Plot of the Runge function (equation (3.4); solid line) and approximations using 10 equidistant points (dashed line) and 20 equidistant points (dashed-dot line).