NA20 Numericka integracija

Ivan Slapničar

4. siječnja 2019.

1 Numerička integracija

1.1 Newton-Cotesove formule

Funkcija $f(x) : [a,b] \to \mathbb{R}$ se interpolira polinomom stupnja n kroz $n+1$ ravnomjerno raspoređenih točaka te se integral aproksimira integralom interpolacijskog polinoma. Polinom možemo računati u Lagrangeovom obliku
(vidi bilježnicu NA09 Interpolacijski polinomi.ipynb):

$$L_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i}$$

Tada je

$$f(x) \approx P_n(x) = \sum_{k=0}^{n} f(x_k) L_k(x),$$

pa je

$$\int_a^b f(x)\,dx \approx \int_a^b P_n(x)\,dx = \sum_{k=0}^{n} f(x_k) \int_a^b L_k(x)\,dx = (b-a) \sum_{k=0}^{n} \omega_k f(x_k). \tag{1}$$

Uz supstituciju $x = a + (b-a)\,t$, *težine* ω_k su

$$\omega_k = \frac{1}{b-a} \int_a^b L_k(x)\,dx = \int_0^1 \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{nt - i}{k - i}\,dt.$$

1.1.1 Trapezna formula

Za $n = 1$ Newton-Cotesova formula (1) daje

$$\omega_0 = \omega_1 = \frac{1}{2}.$$

Interval $[a, b]$ podjelimo na n jednakih podintervala,

$$[x_{i-1}, x_i], \quad i = 1, 2, \ldots, n,$$

i uvedimo oznake

$$\Delta x = \frac{b-a}{n}, \quad y_i = f(x_i).$$

Primijena Newton-Cotesove formule na svaki podinterval i zbrajanje daje *trapeznu formulu*:

$$I_n = \Delta x \left(\frac{y_0}{2} + y_1 + y_2 + \cdots + y_{n-1} + \frac{y_n}{2} \right).$$

Vrijedi

$$\int_a^b f(x)\, dx = I_n + R,$$

pri čemu je *pogreška* R omeđena s

$$|R| \leq \frac{b-a}{12} (\Delta x)^2 \max_{x \in (a,b)} |f''(x)|.$$

Izvod trapezne formule i ocjene pogreške dan je u knjigama Numerička matematika, poglavlje 7.1 i Matematika 2, poglavlje 2.7.2.

1.1.2 Simpsonova formula

Za $n = 2$ Newton-Cotesova formula (1) daje

$$\omega_0 = \frac{1}{6}, \quad \omega_1 = \frac{2}{3}, \quad \omega_2 = \frac{1}{6}.$$

Interval $[a, b]$ podjelimo na paran broj n jednakih podintervala, na svaki podinterval

$$[x_{2i-1}, x_{2i+1}], \quad i = 1, 2, \ldots, \frac{n}{2},$$

primijenimo Newton-Cotesovu formulu i zbrojimo, što daje *Simpsonovu formulu*:

$$I_n = \frac{\Delta x}{3} \left(y_0 + 4(y_1 + y_3 \cdots + y_{n-1}) + 2(y_2 + y_4 + \cdots + y_{n-2}) + y_n \right).$$

Vrijedi

$$\int_a^b f(x)\, dx = I_n + R,$$

pri čemu je *pogreška* R omeđena s

$$|R| \leq \frac{b-a}{180}(\Delta x)^4 \max_{x\in(a,b)} |f^{(4)}(x)|. \tag{2}$$

Za detalje vidi knjige Numerička matematika, poglavlje 7.3 i Matematika 2, poglavlje 2.7.3.

1.1.3 Richardsonova ekstrapolacija

Ocjena pogreške pomoću formula (2) i (3) može biti složena. *Richardsonova ekstrapolacija* nam omogućuje da, uz određene uvjete, pogrešku procijenimo koristeći aproksimaciju integrala s $n/2$ točaka. Ako se u ocjeni pogreške javlja član $(\Delta x)^m$ ($m = 2$ za trapeznu formulu i $m = 4$ za Simpsonovu formulu), tada je pogreška približno manja od broja (vidi Matematika 2, poglavlje 2.7.4)

$$E = \frac{\left(\frac{n}{2}\right)^m}{n^m - \left(\frac{n}{2}\right)^m}(I_n - I_{n/2}).$$

Predznak broja E daje i predznak pogreške, ondosno, ako je $E > 0$, tada je približno

$$\int_a^b f(x)\,dx \in [I_n, I_n + E],$$

a ako je $E \leq 0$, tada je približno

$$\int_a^b f(x)\,dx \in [I_n + E, I_n].$$

```
In [1]: function mytrapez(f::Function,a::Number,b::Number,n::Int64)
            # n je broj intervala
            X=range(a,stop=b,length=n+1)
            Y=map(f,X)
            Δx=(b-a)/n
            I=Δx*(Y[1]/2+sum(Y[2:end-1])+Y[end]/2)
            # Richardsonova ekstrapolacija
            Ihalf=2*Δx*(Y[1]/2+sum(Y[3:2:end-2])+Y[end]/2)
            E=(n/2)^2*(I-Ihalf)/(n^2-(n/2)^2)
            I,E
        end
```

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

```
In [2]: function mySimpson(f::Function,a::Number,b::Number,n::Int64)
            # n je broj intervala, djeljiv s 4
            X=range(a,stop=b,length=n+1)
            Y=map(f,X)
            Δx=(b-a)/n
            I=Δx/3*(Y[1]+4*sum(Y[2:2:end-1])+2*sum(Y[3:2:end-2])+Y[end])
            # Richardsonova ekstrapolacija
```

```
        Ihalf=2*Δx/3*(Y[1]+4*sum(Y[3:4:end-2])+2*sum(Y[5:4:end-4])+Y[end])
        E=(n/2)^4*(I-Ihalf)/(n^4-(n/2)^4)
        I,E
    end
```

```
Out[2]: mySimpson (generic function with 1 method)
```

1.1.4 Primjer 1 - Eliptički integral

Izračunajmo opseg elipse s polu-osima 2 i 1 (vidi Matematika 2, poglavlje 2.7.1). Elipsa je parametarski zadana s

$$x = 2\cos t, \quad y = \sin t, \quad t \in [0, \pi/2]$$

pa je četvrtina opsega jednaka

$$\frac{O}{4} \int_0^{\pi/2} \sqrt{(-2\sin t)^2 + (\cos t)^2}\, dt = 2 \int_0^{\pi/2} \sqrt{1 - \frac{3}{4}(\cos t)^2}\, dt.$$

Radi se o eliptičkom integralu druge vrste koji nije elementarno rješiv, ali se može naći u tablicama. Vidimo da je $O \approx 8 \cdot 1.21125$.

```
In [3]: f1(x)=sqrt(1-(3.0)/4*cos(x)^2)
        mytrapez(f1,0,π/2,4)
```

```
Out[3]: (1.2110515487742433, 0.00036371987130023875)
```

```
In [4]: mytrapez(f1,0,pi/2,10)
```

```
Out[4]: (1.2110560275664024, 1.172757710943273e-7)
```

```
In [5]: mytrapez(f1,0,pi/2,24)
```

```
Out[5]: (1.2110560275684594, 6.439293542825908e-15)
```

```
In [6]: mySimpson(f1,0,π/2,4)
```

```
Out[6]: (1.2114152686455435, -0.000611077902412586)
```

```
In [7]: mySimpson(f1,0,π/2,16)
```

```
Out[7]: (1.2110560276465434, -9.950273232028905e-8)
```

```
In [8]: mySimpson(f1,0,π/2,24)
```

```
Out[8]: (1.211056027568466, -6.556223564047059e-10)
```

1.1.5 Primjer 2 - π

Vrijedi

$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi.$$

Aproksimirajmo π numeričkom integracijom i provjerimo pogrešku (vidi Numerička matematika, poglavlje 7.3).

Pomoću trapezne formula možemo dobiti najviše pet točnih decimala. Simpsonova formula je točnija, ali je konvergencija spora.

```
In [9]: πbig=BigFloat(pi)
```

```
Out[9]: 3.14159265358979323846264338327950288419716939937510582097494459
```

```
In [10]: f2(x)=4/(1+x^2)
         @show πapprox=mytrapez(f2,0,1,10)
         πapprox[1]-πbig
```

```
πapprox = mytrapez(f2, 0, 1, 10) = (3.1399259889071587, 0.0016666250320562053)
```

```
Out[10]: -1.666664682634555812309880018294349956446743125105820974944459e-03
```

```
In [11]: @show πapprox=mytrapez(f2,0,1,100)
         πapprox[1]-πbig
```

```
πapprox = mytrapez(f2, 0, 1, 100) = (3.141575986923129, 1.66666666251795e-5)
```

```
Out[11]: -1.666666666423378282120949501593475335226070323082097494459e-05
```

```
In [12]: @show πapprox=mySimpson(f2,0,1,16)
         πapprox[1]-πbig
```

```
πapprox = mySimpson(f2, 0, 1, 16) = (3.141592651224822, 9.91774099882529e-9)
```

```
Out[12]: -2.364971452347017073514000073718958950156355820974944459e-09
```

```
In [13]: @show πapprox=mySimpson(f2,0,1,64)
         πapprox[1]-πbig
```

```
πapprox = mySimpson(f2, 0, 1, 64) = (3.1415926535892162, 2.4253192047278085e-12)
```

```
Out[13]: -5.769943482751460737218416213332177344808209749445 9e-13
```

1.2 Gaussova kvadratura

Slično kao u formuli (1), integral aproksimiramo sumom umnožaka vrijednosti funkcije i odgovarajućih težina:

$$\int_a^b \omega(x) f(x)\, dx = \sum_{k=1}^{n} \omega_k f(x_k),$$

gdje je $\omega(x)$ neka *težinska funkcija*.

Točke x_k su nul-točke odgovarajućeg ortogonalnog polinoma $P_n(x)$ reda $n+1$, na primjer, *Legendreovih polinoma* za $\omega(x) = 1$ i *Čebiševljevih polinoma* za

$$\omega(x) = \frac{1}{\sqrt{1-x^2}}$$

(vidi bilježnicu NA09 Interpolacijski polinomi.ipynb).

Težine su jednake

$$\omega_k = \int_a^b \omega(x) \prod_{\substack{i=1 \\ i\neq k}}^{n} \frac{x - x_i}{x_k - x_i}\, dx.$$

Pogreška je dana s

$$E = \frac{f^{(2n)}(\xi)}{(2n)!} \int_a^b \omega(x) P_n^2(x)\, dx.$$

Za detalje vidi Numerical Analysis, poglavlje 7.3.

Napomena: Legendreovi i Čebiševljevi polinomi su definirani na intervalu $[-1, 1]$ pa koristimo transformaciju

$$\int_a^b \omega(x) f(x)\, dx = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{b-a}{2} x + \frac{a+b}{2}\right) dx. \tag{3}$$

1.2.1 Postojeće rutine

Profesionalne rutine za numeričku integraciju su složene, a većina programa ima ugrađene odgovarajuće rutine. Tako, na primjer,

- Matlab ima rutinu `quad` koja koristi adaptivnu Simpsonovu formulu, a
- Julia u paketu `QuadGK.jl` ima rutinu `quadgk()` i računa integral u $O(n^2)$ operacija.

Julia također ima i paket `FastGaussQuadrature.jl` koji brzo računa točke i težine za zadani n i razne težinske funkcije pa se pomoću točaka i težina lako izračuna integral u $O(n)$ operacija.

```
In [14]: using QuadGK
```

```
In [15]: ? quadgk

search: quadgk QuadGK
```

```
Out[15]:

quadgk(f, a,b,c...; rtol=sqrt(eps), atol=0, maxevals=10^7, order=7, norm=norm)
```

Numerically integrate the function `f(x)` from `a` to `b`, and optionally over additional intervals `b` to `c` and so on. Keyword options include a relative error tolerance `rtol` (defaults to `sqrt(eps)` in the precision of the endpoints), an absolute error tolerance `atol` (defaults to 0), a maximum number of function evaluations `maxevals` (defaults to `10^7`), and the `order` of the integration rule (defaults to 7).

Returns a pair `(I,E)` of the estimated integral `I` and an estimated upper bound on the absolute error `E`. If `maxevals` is not exceeded then `E <= max(atol, rtol*norm(I))` will hold. (Note that it is useful to specify a positive `atol` in cases where `norm(I)` may be zero.)

The endpoints `a` et cetera can also be complex (in which case the integral is performed over straight-line segments in the complex plane). If the endpoints are `BigFloat`, then the integration will be performed in `BigFloat` precision as well.

> **note**
>
> Note
>
> It is advisable to increase the integration `order` in rough proportion to the precision, for smooth integrands.

More generally, the precision is set by the precision of the integration endpoints (promoted to floating-point types).

The integrand `f(x)` can return any numeric scalar, vector, or matrix type, or in fact any type supporting `+`, `-`, multiplication by real values, and a `norm` (i.e., any normed vector space). Alternatively, a different norm can be specified by passing a `norm`-like function as the `norm` keyword argument (which defaults to `norm`).

> **note**
>
> Note
>
> Only one-dimensional integrals are provided by this function. For multi-dimensional integration (cubature), there are many different algorithms (often much better than simple nested 1d integrals) and the optimal choice tends to be very problem-dependent. See the Julia external-package listing for available algorithms for multidimensional integration or other specialized tasks (such as integrals of highly oscillatory or singular functions).

The algorithm is an adaptive Gauss-Kronrod integration technique: the integral in each interval is estimated using a Kronrod rule (`2*order+1` points) and the error is estimated using an embedded Gauss rule (`order` points). The interval with the largest error is then subdivided into two intervals and the process is repeated until the desired error tolerance is achieved.

These quadrature rules work best for smooth functions within each interval, so if your function has a known discontinuity or other singularity, it is best to subdivide your interval to put the singularity at an endpoint. For example, if `f` has a discontinuity at `x=0.7` and you want to integrate from 0 to 1, you should use `quadgk(f, 0,0.7,1)` to subdivide the interval at the point of discontinuity. The integrand is never evaluated exactly at the endpoints of the intervals, so it is possible to integrate functions that diverge at the endpoints as long as the singularity is integrable (for example, a `log(x)` or `1/sqrt(x)` singularity).

For real-valued endpoints, the starting and/or ending points may be infinite. (A coordinate transformation is performed internally to map the infinite interval to a finite one.)

```
In [16]: quadgk(f1,0,π/2)
```

```
Out[16]: (1.2110560275684594, 8.948231045025068e-11)
```

```
In [17]: quadgk(f2,0,1)
```

```
Out[17]: (3.1415926535897936, 2.6639561667707312e-9)
```

```
In [18]: # Granice mogu biti i beskonačne
         quadgk(x->exp(-x),0,Inf)
```

```
Out[18]: (1.0, 4.5074000326453615e-11)
```

```
In [19]: using FastGaussQuadrature
```

```
In [20]: varinfo(FastGaussQuadrature)
```

```
Out[20]:
```

name	size	summary
FastGaussQuadrature	445.386 KiB	Module
besselroots	0 bytes	typeof(besselroots)
gausschebyshev	0 bytes	typeof(gausschebyshev)
gausshermite	0 bytes	typeof(gausshermite)
gaussjacobi	0 bytes	typeof(gaussjacobi)
gausslaguerre	0 bytes	typeof(gausslaguerre)
gausslegendre	0 bytes	typeof(gausslegendre)
gausslobatto	0 bytes	typeof(gausslobatto)
gaussradau	0 bytes	typeof(gaussradau)

```
In [21]: # Na primjer
         methods(gausschebyshev)
```

```
Out[21]: # 2 methods for generic function "gausschebyshev":
         [1] gausschebyshev(n::Integer) in FastGaussQuadrature at ...
         [2] gausschebyshev(n::Integer, kind::Integer) in FastGaussQuadrature at ...

In [22]: gausschebyshev(16)

Out[22]: ([-0.995185, -0.95694, -0.881921, -0.77301, -0.634393, -0.471397, -0.290285,
         -0.0980171, 0.0980171, 0.290285, 0.471397, 0.634393, 0.77301, 0.881921,
         0.95694, 0.995185], [0.19635, 0.19635, 0.19635, 0.19635, 0.19635, 0.19635,
         0.19635, 0.19635, 0.19635, 0.19635, 0.19635, 0.19635, 0.19635, 0.19635,
         0.19635, 0.19635])

In [23]: # Sada računajmo integrale. U našem slučaju je ω(x)=1
         # pa nam trebaju Legendreovi polinomi
         a=0
         b=π/2
         ξ,ω=gausslegendre(32)
         mapnodes(x)=(b-a)*x/2 .+(a+b)/2

Out[23]: mapnodes (generic function with 1 method)

In [24]: # 1/8 opsega elipse
         using LinearAlgebra
         (b-a)/2*dot(ω,map(f1,mapnodes(ξ)))

Out[24]: 1.2110560275684594

In [25]: # π
         a=0
         b=1
         (b-a)/2*dot(ω,map(f2,mapnodes(ξ)))

Out[25]: 3.141592653589793
```

1.3 Clenshaw-Curtisova kvadratura

Uz supstituciju $x = \cos\theta$, vrijedi

$$I \equiv \int_{-1}^{1} f(x)\,dx = \int_{0}^{\pi} f(\cos\theta)\sin\theta\,d\theta.$$

Integral na desnoj strani se računa integriranjem Fourierovog reda parnog proširenja podintegralne funkcije:

$$I \approx a_0 + \sum_{k=1}^{n} \frac{2a_{2k}}{1-(2k)^2},$$

pri čemu se koeficijenti a_k računaju formulom

$$a_k = \frac{2}{\pi} \int_0^{\pi} f(\cos\theta)\cos(k\theta)\, d\theta.$$

Koeficijenti a_k se mogu računati numeričkom integracijom ili korištenjem brze Fourierove transformacije (FFT), što je puno brže. Za detalje vidi Homer Reid, Clenshaw-Curtis Quadrature.

Ukoliko se integrira na intervalu $[a, b]$, prebacivanje u interval $[-1, 1]$ vrši se kao u formuli (3).

```
In [26]: # Naivna implementacija
         function myClenshawCurtis(f::Function,a::Number,b::Number,n::Int64)
             mapnodes(x)=(b-a)*x/2+(a+b)/2
             z=Vector{Float64}(undef,n)
             g(x)=f(mapnodes(x))
             for i=1:n
                 h(x)=g(cos(x))*cos(2*(i-1)*x)
                 z[i]=2*quadgk(h,0,pi)[1]/pi
             end
             return (z[1]+2*sum([z[i]/(1-4*(i-1)^2) for i=2:n]))*(b-a)/2
         end

Out[26]: myClenshawCurtis (generic function with 1 method)

In [27]: # Implementcija pomoću trapeznog pravila s m podintervala
         function myClenshawCurtis(f::Function,a::Number,b::Number,n::Int64,m::Int64)
             mapnodes(x)=(b-a)*x/2+(a+b)/2
             z=Vector{Float64}(undef,n)
             g(x)=f(mapnodes(x))
             ls=range(0,stop=pi,length=m+1)
             u=map(x->g(cos(x)),ls)
             for i=1:n
                 v=map(x->cos(2*(i-1)*x),ls)
                 z[i]=(u[1]*v[1]+2*(u[2:end-1]⋅v[2:end-1])+v[end]*u[end])/m
             end
             return (z[1]+2*sum([z[i]/(1-4*(i-1)^2) for i=2:n]))*(b-a)/2
         end

Out[27]: myClenshawCurtis (generic function with 2 methods)

In [28]: myClenshawCurtis(f1,0,pi/2,8)

Out[28]: 1.2110560274835651

In [29]: myClenshawCurtis(f1,0,pi/2,8,50)

Out[29]: 1.2110560274835651
```

```
In [30]: myClenshawCurtis(f2,0,1,16,50),pi
```

```
Out[30]: (3.1415926535897936, π = 3.1415926535897...)
```

```
In [31]: myClenshawCurtis(x->exp(-x),0,1000,100,200)
```

```
Out[31]: 1.0000000000000009
```

```
In [32]: # Najbrža implementacija pomoću fft(), 2^n je broj točaka
         using FFTW
         function myClenshawCurtis(f::Function,a::Number,b::Number,n::Int64)
             mapnodes(x)=(b-a)*x/2+(a+b)/2
             g(x)=f(mapnodes(x))
             w=map(x->g(cos(x)),range(0,stop=2*pi,length=2^n))
             w[1]=(w[1]+w[end])/2
             z=real(fft(w))
             z/=2^(n-1)
             return (z[1]+2*sum([z[i]/(1-(i-1)^2) for i=3:2:2^(n-1)]))*(b-a)/2
         end
```

```
Out[32]: myClenshawCurtis (generic function with 2 methods)
```

```
In [33]: myClenshawCurtis(f1,0,pi/2,11)
```

```
Out[33]: 1.2113091816227997
```

```
In [34]: myClenshawCurtis(f2,0,1,16),pi
```

```
Out[34]: (3.141571198232438, π = 3.1415926535897...)
```