INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Číslo projektu	CZ.1 .07/1.1 .36/02.0066
Autor	Ladislav Kašpárek
Předmět	Programovací metody
Téma	Numerická integrace
Metodický pokyn	výkladový text s ukázkami, experimentální dokument

Numerická integrace

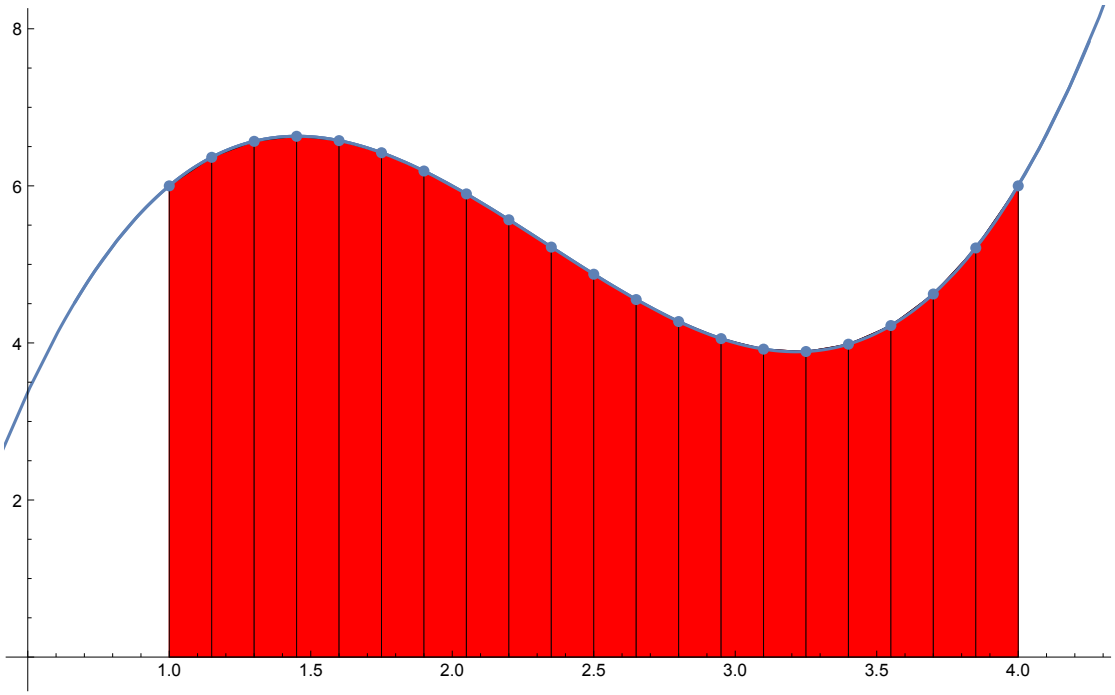
Nyní můžeme přistoupit k vlastnímu výpočtu plochy pod křivkou. Další metoda se nazývá lichoběžníková, dává velmi dobré výsledky integrace a v parxi se často používá. Výhodou této metody je její jednoduchost, rychlost a doprá přesnost. Zpravidla její výsledky postačují a není nutné používat další metody nebo vylepšení.

Lichoběžníková metoda

Lichoběžníková metoda vychází z myšlenky rozdělit interval <a, b> na dílky (lichoběžníky) a jejich plochy sečíst. Viz následující obrázek:

In[1]:=
```
f[x_] = (x - 4) (x - 1) (x - 2) + 6;
body = Table[{1 + nn * (3 / 20), f[1 + nn * (3 / 20)]}, {nn, 0, 20}];
gr = Graphics[{EdgeForm[Black], Red,
    Table[Polygon[{{1 + nn * (3 / 20), 0}, {1 + nn * (3 / 20), f[1 + nn * (3 / 20)]},
       {1 + (nn + 1) * (3 / 20), f[1 + (nn + 1) * (3 / 20)]},
       {1 + (nn + 1) * (3 / 20), 0}}], {nn, 0, 19}]}];
Show[Plot[f[x], {x, 2 / 3, 4 + 1 / 4}, AxesOrigin → {1 / 2, 0}],
 Plot[f[x], {x, 1, 4}, Filling → {1 → {0, Lighter[Red, .7]}}],
 gr, ListPlot[body, PlotStyle → PointSize[.01]],
 Plot[f[x], {x, 0, 5}], ImageSize → Large]
```

Out[4]=



Jak již bylo naznačeno, interval <a, b> je rozdělen na dílky a sestaven lichoběžník se šířkou dílku a délkami stran podle funkčních hodnot na pravém a levém bodu dílku. Vypočítat obsah lichoběžníku je snadné: šířka je dána a výška je průměrem levé a pravé funkční hodnoty. To je zásadní rozdíl proti obdélníkům, kde se bere jako výška jen hodnota jediná (kdekoliv z šířky dílku).
Při bližším pohledu na obrázek lze vysledovat, že původní křivka je nahrazena “jakousi” lomenou čárou sestavenou z úseček. Lomená čára lépe charakterizuje průběh funkce proti obdélníkové metodě.

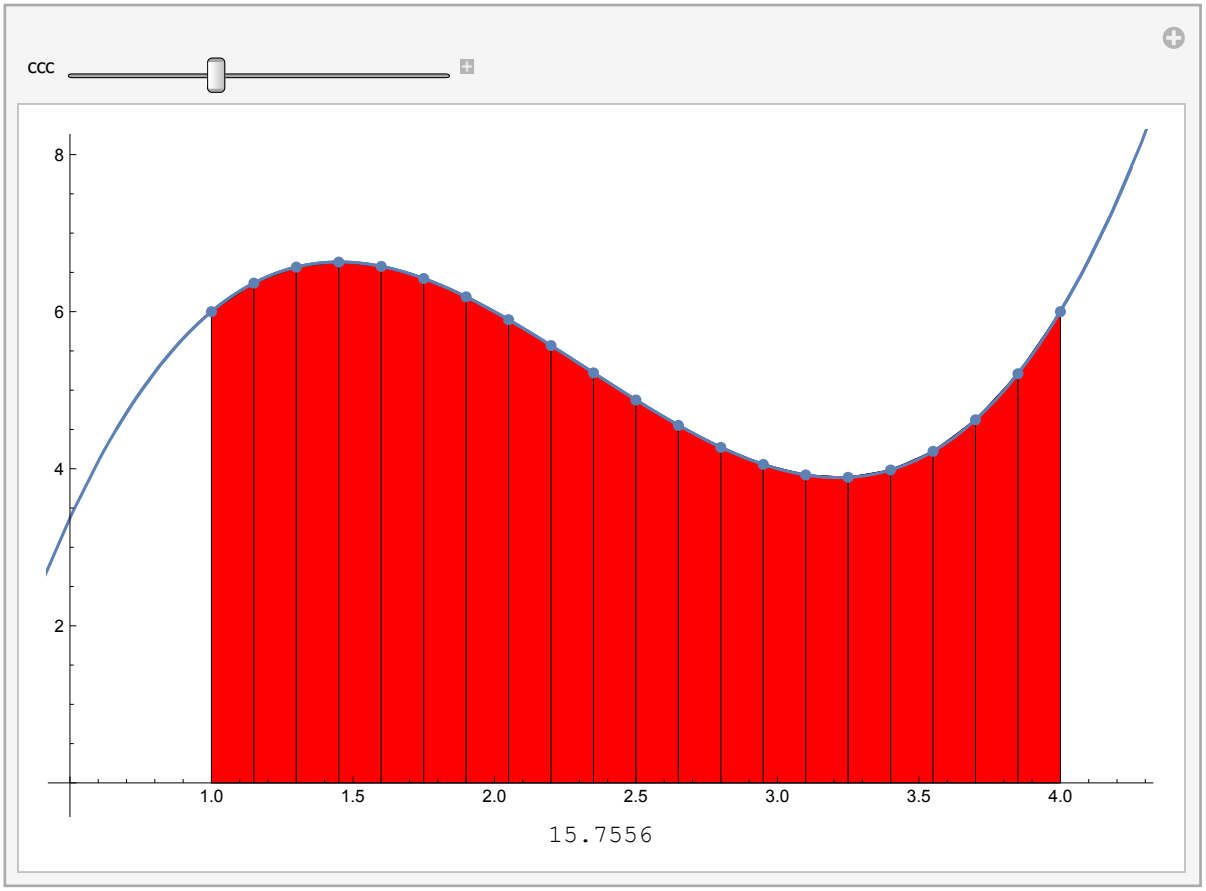
Přesnost výpočtu

Podobně jako u obdélnikové metody přesnost výpočtu ovlivňují zejména dva faktory: sama funkce, ale tu těžko ovlivníme a také počet dílků na které interval rozdělíme. Teoreticky čím větší počet dílků, tím větší přesnost.
Ze vzorce pro odhad chyby je možno stanovit přibližný počet dílků (segmentů) stejné délky, při zadané maximální povolené chybě. Odhad chyby:

$$\frac{\max_{\langle a,b\rangle} |f''(x)|}{12\,n^2}\,(b-a)^3$$

In[5]:=
```
Manipulate[body = Table[{1 + nn * (3 / (ccc)), f[1 + nn * (3 / (ccc))]}, {nn, 0, ccc}];
 lich = Table[Polygon[{{1 + nn * (3 / ccc), 0}, {1 + nn * (3 / ccc), f[1 + nn * (3 / ccc)]},
      {1 + (nn + 1) * (3 / ccc), f[1 + (nn + 1) * (3 / ccc)]},
      {1 + (nn + 1) * (3 / ccc), 0}}], {nn, 0, ccc - 1}];
 gr = Graphics[{EdgeForm[Black], Red, lich}];
 Column[{Show[Plot[f[x], {x, 2 / 3, 4 + 1 / 4}, AxesOrigin → {1 / 2, 0}],
    Plot[f[x], {x, 1, 4}, Filling → {1 → {0, Lighter[Red, .7]}}], gr,
    ListPlot[body, PlotStyle → PointSize[.01]], Plot[f[x], {x, 0, 5}],
    ImageSize → Large], Map[Area, lich] // Total // N}, Center],
 {{ccc, 20}, 2, 50, 1}, SaveDefinitions → True]
```

Out[5]=



Pokusme se pomocí vzorce odhadnout počet dílků pro chybu pod hranicí 1/1000. Protože předpokládáme, že čtenář neumí derivovat, tak druhou derivaci naší funkce nalezne system *Mathematica* a její maximum na daném intervalu také:

In[6]:=
```
f''[x]
```

Out[6]=
```
-14 + 6 x
```

In[7]:=
```
Solve[1/(12 n^2) (FindMaximum[{Abs[f''[x]], 1 ≤ x ≤ 4}, x] // First) (4 - 1)^3 == 0.001, n]
```

Solve::ratnz : Solve was unable to solve the system with inexact coefficients. The answer was obtained by solving a corresponding exact system and numericizing the result. >>

Out[7]=
```
{{n → -134.164}, {n → 134.164}}
```

Rovnice dává dva kořeny, záporný můžeme vyloučit. Výsledek je tedy 134, tedy **nejhůře** pro 135

dílků a více bude chyba pod hranicí jedné tisíciny. Protože se jedná o nejhorší možný odhad, je možné, že požadované přesnosti dosáhneme i při menším počtu dílků, ale 135 je jistota.

Další metody

V literatuře je často uváděna ještě jedna numerická metoda pro integraci, tzv Simpsonovo pravidlo. Tato metoda nahrazuje původni funkci (křivku) na daném “dvoudílku” nikoliv úsečkou, ale částí paraboly. Tyto paraboly ještě lépe vystihují průběh původní funkce a dochází k menším chybám. ALE výpočet je o něco složitější a trvá déle, ALE zase je možné zmenšit počet dílků pro zadanou přesnost. Otázkou zůstává zda-li se vyplatí tuto metodu použít místo metody lichoběžníkové...

Další metody jsou založeny na myšlence automatické volby kroku. Tedy metota (ať už používá obdélníky nebo lichoběžníky) si automaticky volí šířku dílku. Je-li “změna” na křivce malá volí se krok širší (a tím je výpočet rychlejší bez ztráty přesnosti), nebo mění-li křivka svůj průběh hodně-prudce, volí se krok menší, jemnější, aby byla přesnost zachována. Otázkou zůstává zda-li se vyplatí tuto metodu použít místo metody lichoběžníkové...

Kód metody v jazyce JAVA

```
/* *
  * Auto Generated Java Class.
  */
  public class Integral {

    public static double f (double x) {
       return x*x*x - 7*x*x + 14*x - 2;
     }

    public static double lichobeznik (double a, double b, long dilek) {
       double suma = 0;
       final double DELTAX = (b - a)/dilek;

       for (int i = 0; i < dilek; i++) {
          suma = suma + (DELTAX*(   (f (a + i*DELTAX) + f (a + (i + 1)*DELTAX))/2   ));
        }
       return suma;
     }

    public static void main (String[] args) {
       System.out.print (  lichobeznik (1, 4, 3000)  );
     }
  }
```

Shrnutí

Lichoběžníková metoda je přesnější a efektivnější než metoda obdélníková, známe i odhad chyby. Animace nám ukaáže jak přesnost výpočtu závysí na počtu dílků nad intervalem. Je zde uveden i kód v jazyce Java.

Sbírka úloh

1/ Sestavte tabulku velikostí chyb pro 10, 15, 20, 30, 50, 100 a 150 dílků.
2/ Kolik dílků je potřeba zvolit pro přesnost pod hranicí 13/55555?
3/ Integrujte funkci sinus na intervalu <0, 1> porovnejte výsledek s obdélníkovou metodou. Experimentujte se stejnými počty dílků, nebo pokuste se najít počet dílků aby byla obdélníková metoda stejná jako lichoběžníková.
4/ Integruj funkce nad intervalem <0, 1> : $\sqrt{x}$, x^2, x^3, sin(x), cos(x), log(x), 10^x. Experimentuj s počtem dílků při integraci. Porovnej výsledky z Java programu s výsledky Wolfram Mathamatici.
5/ Experimentuj s vysokým počtem dílků v jazyce Java na intervalem <a, b>. Prvně nech výpočty běžět v datovém typu float a pak v double. Sleduj přesnost výpočtu a jeho délku/čas.
6/ Optimalizujte kód v jazyce Java, tak aby se funkční hodnoty nepočítaly 2x.

Zdroje

http://www.kvd.zcu.cz/cz/materialy/numet/_numet.html#_Toc501178915
http://reference.wolfram.com/mathematica/ref/Integrate.html
http://reference.wolfram.com/mathematica/ref/NIntegrate.html
http://www.matematika.cz/urcity-integral