Zaawansowane metody numeryczne

W. Hebisch

2 marca 2022

1 Program i literatura

Literatura

[1] S. C. Brenner, L. R. Scott, The Mathematical Theory of Finite Element Methods, Springer, 2008.

[2] E. Hairer, S. P. Norsett, and G. Wanner, Solving ordinary differential equations, I: Nonstiff problems, Springer, 1993.

[3] E. Hairer, G. Wanner, Solving ordinary differential equations II: Stiff and Differential - Algebraic Problems, Springer, 1996.

[4] D. V. Hutton, Fundamentals of Finite Element Analysis, dostępna w sieci.

[5] A. Ralston, Wstęp do analizy numerycznej, PWN, Warszawa 1983.

[6] A.A.Samarski, J.S.Nikołajew, Metody rozwiązywania równań siatkowych, PWN.

- Metody całkowania numerycznego wysokiego rzędu, błąd metody.
- Arytmetka zmiennopozycyjna i błędy zaokrągleń.
- Numeryczne rozwiązywanie równań różniczkowych zwyczajnych: podstawowe metody, szacowanie błędu, rząd metody, badanie zbieżności, stabilność.
- Funkcje sklejane i interpolacja wielowymiarowa, komputerowa reprezentacja funkcji.
- Rówania różniczkowe cząstkowe:
  - Proste metody rozwiązywania.
  - Rozwinięcia na funkcje własne, użycie FFT.
  - Metody wielosiatkowe (multigrid)
  - Metoda elementów skończonych, metody wariacyjne.

2 System FriCAS

Cechy systemu:

- Często wymaga jawnych deklaracji typów.
- Wcięcia są istotne, ciąg lini z takimi samymi wcięciami tworzy blok (instrukcję złożoną).
- Typy `DoubleFloat`, `DoubleFloatVector` i `DoubleFloatMatrix` są kompilowane to kodu maszynowego i dają stosunkowo dobrą wydajność.
- Typ `Float` daje arytmetykę zmiennopozycyjną dowolnej precyzji.
- Obliczenia symboliczne.

System jest zainstalowny w pracowniach, można się do niego dostać przy pomocy `ssh`. Po uruchomieniu pod Linuxem w środowisku graficznym pokazuje się interaktywna pomoc i dokumentacja.

System można pobrać ze strony projektu `fricas.sf.net`

Dostępne funkcje moża przeglądać przez interfejs sieciowy `http://fricas.github.io/api`

Podstawowa dokumentacja to FriCAS Book, dostępna z pomocy interaktywnej, a także jako `.pdf` z `http://fricas.github.io/book.pdf`, dla na najbardziej istotny jest rozdział 2.

3 Całkowanie numeryczne

3.1 Experyment numeryczny

Popatrzmy teraz na 3 proste (i niezbyt dobre) funkcje całkujące:

```
int1(f, n) ==
   s := 0.0
   h := 1.0/(n::Float)
   for i in 0..n repeat
       s := s + f(i*h)
   s/(n+1)::Float

int2(f, n) ==
   s := 0.0
   h := 1.0/(n::Float)
   for i in 1..n repeat
       s := s + f(i*h)
   h*s

int3(f, n) ==
   s := 0.5*(f(0.0) + f(1.0))
   h := 1.0/(n::Float)
   for i in 1..(n-1) repeat
       s := s + f(i*h)
   h*s
```

Odpowiadają im odpowiednio wzory:

$$\int_0^1 f(x)dx \approx \frac{1}{n+1}\sum_{i=0}^{n} f(\frac{i}{n})$$

$$\int_0^1 f(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} f(\frac{i}{n})$$

$$\int_0^1 f(x)dx \approx \frac{1}{n}\left(\frac{1}{2}(f(0)+f(1)) + \sum_{i=1}^{n-1} f(\frac{i}{n})\right)$$

Można je użyć następująco

```
f : Float -> Float
f := x +-> x + 1

int1(f, 10)
int2(f, 10)
int3(f, 10)

f1(x : Float) : Float == sin(x)
v1 := 1.0 - cos(1.0)

[int1(f1, 10*k) - v1 for k in 1..10]
[int2(f1, 10*k) - v1 for k in 1..10]
[int3(f1, 10*k) - v1 for k in 1..10]

f2(x : Float) : Float == x*(1-x)
v2 := (1/6)::Float

[int1(f2, 10*k) - v2 for k in 1..10]
[int2(f2, 10*k) - v2 for k in 1..10]
[int3(f2, 10*k) - v2 for k in 1..10]
```

Dostajemy wtedy następujący wynik:

```
f := x +-> x + 1


   (5)  theMap(*1;anonymousFunction;0;frame1;internal)
                                                       Type: (Float -> Float)

int1(f, 10)

   Compiling function int1 with type ((Float -> Float), PositiveInteger
      ) -> Float
```

```
   (6)  1.5
                                                                   Type: Float
int2(f, 10)

   Compiling function int2 with type ((Float -> Float), PositiveInteger
      ) -> Float

   (7)  1.55
                                                                   Type: Float
int3(f, 10)

   Compiling function int3 with type ((Float -> Float), PositiveInteger
      ) -> Float

   (8)  1.5
                                                                   Type: Float

f1(x : Float) : Float == sin(x)

   Function declaration f1 : Float -> Float has been added to
      workspace.
                                                                    Type: Void
v1 := 1.0 - cos(1.0)


   (10)  0.4596976941_318602826
                                                                   Type: Float

[int1(f1, 10*k) - v1 for k in 1..10]

   Compiling function f1 with type Float -> Float

   (11)
   [- 0.0038903322_2425014601, - 0.0019465566_000082402,
    - 0.0012980375_295094685, - 0.0009736564_6911171445,
    - 0.0007789876_4037776338, - 0.0006491913_7218744678,
    - 0.0005564713_0632433202, - 0.0004869266_0172403901,
    - 0.0004328335_0128367584, - 0.0003895572_6513028831]
                                                             Type: List(Float)
[int2(f1, 10*k) - v1 for k in 1..10]


   (12)
   [0.0416904039_665108676, 0.0209410002_765843619, 0.0139819510_239022253,
    0.0104944444_724569998, 0.0083993864_89451887, 0.0070016170_0714043382,
    0.0060026890_1975475299, 0.0052532079_9240266403, 0.0046701093_9461161979,
```

```
    0.0042035241_0353701163]
                                                     Type: List(Float)
[int3(f1, 10*k) - v1 for k in 1..10]


   (13)
   [- 0.0003831452_7388395769, - 0.0000957743_4361305075,
    - 0.0000425653_8956271646, - 0.0000239428_376417066,
    - 0.0000153233_586270781, - 0.0000106411_995920371,
    - 0.0000078180_1458736491, - 0.0000059856_6264668914,
    - 0.0000047294_0987669413, - 0.0000038308_205024709]
                                                     Type: List(Float)

f2(x : Float) : Float == x*(1-x)

   Function declaration f2 : Float -> Float has been added to
      workspace.
                                                            Type: Void
v2 := (1/6)::Float


   (15)  0.1666666666_6666666667
                                                           Type: Float
[int1(f2, 10*k) - v2 for k in 1..10]

   Compiling function f2 with type Float -> Float

   (16)
   [- 0.0166666666_6666666667, - 0.0083333333_3333333333,
    - 0.0055555555_5555555556, - 0.0041666666_6666666667,
    - 0.0033333333_3333333333, - 0.0027777777_7777777778,
    - 0.0023809523_8095238095, - 0.0020833333_3333333333,
    - 0.0018518518_5185185185, - 0.0016666666_6666666666]
                                                     Type: List(Float)
[int2(f2, 10*k) - v2 for k in 1..10]


   (17)
   [- 0.0016666666_6666666667, - 0.0004166666_6666666667,
    - 0.0001851851_8518518518, - 0.0001041666_666666667,
    - 0.0000666666_6666666667, - 0.0000462962_962962963,
    - 0.0000340136_0544217687, - 0.0000260416_6666666667,
    - 0.0000205761_316872428, - 0.0000166666_6666666666]
                                                     Type: List(Float)
[int3(f2, 10*k) - v2 for k in 1..10]
```

```
(18)
[- 0.0016666666_6666666667, - 0.0004166666_6666666667,
 - 0.0001851851_8518518518, - 0.0001041666_666666667,
 - 0.0000666666_6666666667, - 0.0000462962_962962963,
 - 0.0000340136_0544217687, - 0.0000260416_6666666667,
 - 0.0000205761_316872428, - 0.0000166666_6666666666]
                                                         Type: List(Float)
```

Uwaga: cały przykład jest w pliku `ex_int.input` i można go wykonać poleceniem `)read` tj:

```
)read ex_int.input
```

Aby dokładniej zbadać zachowanie błędu możemy użyć program przykładowy `sqr_fit.input`

Teoria przewiduje że błąd powinien maleć proporcjonalnie do odwrotności pewnej potęgi liczby kroków. Wykładnik tej potęgi nazywamy rzędem metody. Interesuje nas też współczynnik, większy współczynnik oznacza większy błąd. Wywołanie `sqr_fit(lx, ly , k)` znajduje najlepsze dopasowanie średnikwadratowe ciągu wartości `ly` w punktach `lx` do kombinacji liniowej $1, x^{-1}, \ldots, x^{-k}$. Ze względu na błąd zaokrąglenia nawet tam gdzie teoria przewiduje 0 pojawią się małe, ale niezerowa wartości. Dlatego w praktyce bardzo małe współczynniki należy pomijać, a istotny jest najmniejsze wykładnik z niezbyt małym współczynnikiem.

Uwaga: trzeci parameter podaje maksymalny co do wartości bezwzględnej wykładnik x^{-k}, duże wartości są niezbyt przydatne ze względu na ograniczoną dokładność obliczeń.

`sqr_fit` możemy użyć następująco (komendy w pliku `int_fits.input`):

```
sqr_fit([10*k for k in 1..10], [int1(f1, 10*k) - v1 for k in 1..10], 4)
sqr_fit([10*k for k in 1..10], [int2(f1, 10*k) - v1 for k in 1..10], 4)
sqr_fit([10*k for k in 1..10], [int3(f1, 10*k) - v1 for k in 1..10], 4)
sqr_fit([10*k for k in 1..10], [int1(f2, 10*k) - v2 for k in 1..10], 4)
sqr_fit([10*k for k in 1..10], [int2(f2, 10*k) - v2 for k in 1..10], 4)
sqr_fit([10*k for k in 1..10], [int3(f2, 10*k) - v2 for k in 1..10], 4)
```

Wyniki:

```
sqr_fit([10*k for k in 1..10], [int1(f1, 10*k) - v1 for k in 1..10], 4)


   (42)
                                   4                                   3
     0.0000123154_0114968611_1385 x_inv  - 0.0006538237_9763016718_862 x_inv
   +
                                  2
     0.0006540531_0458445478_574 x_inv  - 0.0389622016_2541092691_1 x_inv
   +
     - 0.4973567723_81509263 E -12
                                                      Type: Polynomial(Float)
sqr_fit([10*k for k in 1..10], [int2(f1, 10*k) - v1 for k in 1..10], 4)
```

```
   Compiling function int2 with type ((Float -> Float), PositiveInteger
      ) -> Float

   (43)
                                         4                                  3
     - 0.0006389133_8837126142_146 x_inv  + 0.4169138284_470592 E -7 x_inv
   +
                                       2
     - 0.0383081426_2156875704_5 x_inv  + 0.4207354924_2477705513 x_inv
   +
     - 0.1036901243_59538 E -12
                                                      Type: Polynomial(Float)
sqr_fit([10*k for k in 1..10], [int3(f1, 10*k) - v1 for k in 1..10], 4)


   (44)
                                         4                                  3
     - 0.0006389133_8862988506_207 x_inv  + 0.4169143062_586208 E -7 x_inv
   +
                                       2
     - 0.0383081426_2157148136_4 x_inv  + 0.2082885869_403825 E -10 x_inv
   +
     - 0.1036904916_48812 E -12
                                                      Type: Polynomial(Float)
sqr_fit([10*k for k in 1..10], [int1(f2, 10*k) - v2 for k in 1..10], 4)


   (45)
                                          4
     - 0.9546030481_532350159 E -13 x_inv
   +
                                         3                                        2
     0.1760530163_8744552644 E -13 x_inv  - 0.9999693181_1530542364 E -15 x_inv
   +
     - 0.1666666666_6666664596 x_inv - 0.1318711939_4196428242 E -18
                                                      Type: Polynomial(Float)
sqr_fit([10*k for k in 1..10], [int2(f2, 10*k) - v2 for k in 1..10], 4)


   (46)
                                           4
     - 0.8035266142_9962486564 E -14 x_inv
   +
                                         3                                  2
     0.1459358437_8775913009 E -14 x_inv  - 0.1666666666_6666674722 x_inv
   +
```

```
    0.1577713197_7935425617 E -17 x_inv - 0.8399286016_6035550035 E -20
                                                 Type: Polynomial(Float)
sqr_fit([10*k for k in 1..10], [int3(f2, 10*k) - v2 for k in 1..10], 4)


   (47)
                                        4
    - 0.8035266142_9962486564 E -14 x_inv
   +
                                      3                                 2
    0.1459358437_8775913009 E -14 x_inv  - 0.1666666666_6666674722 x_inv
   +
    0.1577713197_7935425617 E -17 x_inv - 0.8399286016_6035550035 E -20
                                                 Type: Polynomial(Float)
```

Wyniki wskazują że dla pierwszej funkcji tzn. $\cos(x)$ pierwsza i druga metoda ma rząd 1, lecz pierwsza ma mniejszy współczynnik błedu. Trzecia metoda ma rząd 2.

Dla drugiej funkcji pierwsza metoda dalej ma rząd 1, zaś druga i trzecia ma rząd 2. Jak się przypatrzymy dokładniej, w tym przypadku druga i trzecia metoda dają ten sam wynik: metody różnią się tylko tym jak traktujemy punkty na końcach przedziału, a tam druga funkcja ma wartość 0.

3.2 Analiza

Przypomnijmy wzór trzeciej metody:

$$\int_0^1 f(x)dx \approx \frac{1}{n}\left(\frac{1}{2}(f(0)+f(1)) + \sum_{i=1}^{n-1} f(\frac{i}{n})\right)$$

Ten wzór można interpretować następująco: przedział $[0,1]$ dzielimy na n podprzedziałów $[0,1/n]$, $[1/n,2/n]$, $\ldots$, $[(n-1)/n,1]$. Na każdym z podprzedziałów stosujemy wzór:

$$\int_a^b f(x)dx \approx \frac{b-a}{2}(f(a)+f(b)).$$

Analizę wygodniej przeprowadzić dla $a=-1$, $b=1$. Jako pierwszy krok patrzymy na różnicę:

$$\int_0^1 f(x)dx - f(0).$$

Lemat 3.1

$$\int_0^1 f(x)dx - f(0) = \int_0^1 (1-s)f'(s)ds = \frac{1}{2}(f'(0) + \int_0^1 (1-s)^2 f''(s)ds)$$

Dowód:

$$f(x) = f(0) + \int_0^x f'(s)ds$$

czyli

$$\begin{aligned}\int_0^1 f(x)dx - f(0) &= \int_0^1 \int_0^x f'(s)dsdx \\ &= \int_0^1 f'(s)\int_s^1 dxds = \int_0^1 (1-s)f'(s)ds \\ &= \frac{1}{2}(f'(0) + \int_0^1 (1-s)^2 f''(s)ds)\end{aligned}$$

gdzie ostatni krok to całkowanie przez części. $\square$

Lemat 3.2

$$\int_0^1 f(x)dx - f(1) = \frac{1}{2}(-f'(0) - \int_0^1 (1-s^2)f''(s)ds)$$

Dowód: Na mocy poprzedniego lematu

$$f(1) - f(0) = \int_0^1 f'(x)dx = \int_0^1 (1-s)f''(s)ds + f'(0)$$

Odejmując od wzoru z poprzedniego lematu mamy:

$$\begin{aligned}\int_0^1 f(x)dx - f(1) &= \frac{1}{2}(f'(0) + \int_0^1 (1-s)^2 f''(s)ds) - \int_0^1 (1-s)f''(s)ds - f'(0) \\ &= \frac{1}{2}(-f'(0) + \int_0^1 ((1-s)^2 - 2(1-s))f''(s)ds)\end{aligned}$$

Dalej

$$(1-s)^2 - 2(1-s) = 1 - 2s + s^2 - 2 + 2s = -1 + s^2 = -(1-s^2)$$

co daje wynik. $\square$

Teraz

$$\begin{aligned}\int_{-1}^1 f(x)dx - (f(1) + f(-1)) &= \int_{-1}^0 f(x)dx - f(-1) + \int_0^1 f(x)dx - f(1) \\ &= \int_0^1 f(-x)dx - f(-1) + \int_0^1 f(x)dx - f(1) \\ &= \frac{1}{2}(f'(0) - \int_0^1 (1-s^2)f''(-s)ds) + \frac{1}{2}(-f'(0) - \int_0^1 (1-s^2)f''(s)ds) \\ &= -\frac{1}{2}(\int_0^1 (1-s^2)^2 f''(-s)ds - \int_0^1 (1-s^2)f''(s)ds) \\ &= -\frac{1}{2}\int_{-1}^1 (1-x^2)f''(x)dx\end{aligned}$$

co daje błąd dla trzeciej metody na pojedynczym odcinku.

Następnie, przy przejściu na odcinek o długości h całki mnożą się przez ten sam czynik, zaś druga pochodna mnoży się przez $(h/2)^2$, czyli

$$\int_a^{a+h} f(x)dx - \frac{h}{2}(f(a) + f(a+h)) = -\frac{h^2}{8}\int_a^{a+h}(1-(x-a-h/2)^2)f''(x)dx$$

Przy podziale na odcinki błędy się dodają, czyli błąd można zapisać w postaci

$$\int_a^{a+nh} f(x)dx - \frac{h}{2}\sum_{i=0}^{n}(f(a+ih) + f(a+(i+1)h) = -\frac{h^2}{8}\int_a^{a+nh}\omega(x)f''(x)dx$$

gdzie na poszczególnych odcinkach podziału ω jest zadana przez poprzedni wzór. Dla n dażącego do nieskończoności i ciągłej f'' mamy

$$\lim_{n\to\infty}\int_a^b \omega(x)f''(x)dx = \frac{2}{3}\int_a^b f''(x)dx = \frac{2}{3}(f'(b) - f'(a))$$

Ogólniej wzór całkowy pokazuje że dla regularnych funkcji błąd jest gładką funkcją parametrów.

Pierwsza i druga metoda różnią się od trzeciej o człony na końcach przedziału, które wprowadzają błąd rzedu h, czyli w przeciwieństwie do metody trzeciej są one rzędu 1. Jeśli wartości funkcji na końcach przedziału są równe to druga metoda daje taki sam wynik jak trzecia, czyli też rząd 2. Dalej się okaże że dla funkcji okresowych druga i trzecia metoda są nieskończonego rzędu, tak że błąd maleje bardzo szybko z krokiem.

4 Dodatek

Poniżej podajemy treść programu `sqr_fit.input`:

```
-- dF ==> DoubleFloat
dF ==> Float
vF ==> Vector(dF)
lF ==> List(dF)
gR ==> Record(base : List(vF), coeffs : Matrix(dF))

gramm_schmidt(lv : List(vF)) : gR ==
    res : List(vF) := []
    n := #lv
    tm := zero(n, n)$Matrix(dF)
    for v in lv for i in 1..n repeat
        for bv in res for k in (i - 1)..1 by -1 repeat
            ck := dot(v, bv)
            v := v - ck*bv
            tm(i, k) := ck
        nv := sqrt(dot(v, v))
        res := cons((1/nv)*v, res)
```

```
        tm(i, i) := nv
    [reverse!(res), tm]

pF ==> Polynomial(dF)
sqr_fit(lx : lF, ly : lF, k : Integer) : pF ==
    n := #lx
    bl : List(vF) := []
    for i in 0..k repeat
        v := zero(n)$vF
        for xj in lx for j in 1..n repeat
            v(j) := xj^(-i)
        bl := cons(v, bl)
    bl := reverse!(bl)
    gr := gramm_schmidt(bl)
    -- print(gr)
    bl := gr.base
    tm := inverse(gr.coeffs)
    vy : vF := vector(ly)
    resv : vF := zero((k+1)::NonNegativeInteger)
    for i in 0..k for bv in bl repeat
        ci := dot(vy, bv)
        resv := resv + ci*row(tm, i+1)
    res : pF := 0
    for i in 0..k repeat
        res := res + monomial(resv(i+1)::pF, 'x_inv, i::NonNegativeInteger)
    res
```