Numeryczne rozwiązywanie równań różniczkowych zwyczajnych

Paweł Jankowski Jakub Narębski

8 marca 2001 roku

1 Wstęp

Rozwiązywanie równań różniczkowych zwyczajnych można zawsze sprowadzić do rozwiązywania układu równań rózniczkowych pierwszego rzędu. Najprościej jest wybrać nowe zmienne tak, by były po prostu pochodnymi. Wystarczy zatem rozważać układ równań różniczkowych pierwszego rzędu. Rozpatrywać będziemy *zagadnienia początkowe*, tzn. będziemy chcieli znaleźć rozwiązanie równania o zadanej wartości $\mathbf{u}_0$ w punkcie początkowym t_0:

$$\begin{cases} \dfrac{d\mathbf{u}(t)}{dt} = \mathbf{f}(\mathbf{u}, t) \\ \mathbf{u}(t_0) = \mathbf{u}_0 \end{cases} \tag{1}$$

Rozważać będziemy *metody różnicowe*, które wyznaczają przybliżenia jedynie wartości rozwiązania $\mathbf{u}(t_i)$ w pewnych punktach t_i. Będzimy się zajmowali przede wszystkim przypadkiem, gdy punkty t_i są rozmieszczone równomiernie co Δt, tzn. $t_i = t_0 + i\Delta t$. Wielkość Δt nazywamy *krokiem całkowania*. Metody róznicowe dają nam wzór na wyliczenie wartości funkcji w kroku następnym t_{n+1} na podstawie kroków poprzednich.

Dla uproszczenia wprowadzimy następujące oznaczenia

$$t_i = t_0 + i\,\Delta t \tag{2a}$$
$$\mathbf{u}_i = \mathbf{u}(t_i) \tag{2b}$$
$$\mathbf{f}_i = \mathbf{f}(\mathbf{u}_i, t_i) \tag{2c}$$

Ogólną metodą na wyprowadzenie metod różnicowych rozwiązywania układu równań różniczkowych zwyczajnych jest zapisanie go w postaci całkowej

i zastosowanie przybliżenia na całkę po prawej stronie

$$\frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u}(t), t)$$
$$\int_{t_n}^{t_{n+1}} \frac{d\mathbf{u}}{dt}\, dt = \int_{t_n}^{t_{n+1}} \mathbf{f}(\mathbf{u}(t), t)\, dt$$
$$\mathbf{u}(t_{n+1}) - \mathbf{u}(t_n) = \int_{t_n}^{t_{n+1}} \mathbf{f}(\mathbf{u}(t), t)\, dt$$

2 Metoda Eulera

Najprostszą metodą różnicową jest metoda Eulera. Pochodną $\mathbf{u}'(t)$ przybliżamy ilorazem różnicowym pierwszego rzędu w przód opartym na węzłach t_n i $t_{n+1} = t_n + \Delta t$. Inaczej: przybliżamy całkę za pomocą wartości funkcji w początku przedziału razy jego długość:

$$\int_{t_n}^{t_{n+1}} \mathbf{f}(\mathbf{u}(t), t)\, dt \simeq \Delta t\, \mathbf{u}(t_n)$$

Dostajemy zatem następujący wzór na wartość rozwiązania w kroku następnym

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \Delta t\, \mathbf{f}(\mathbf{u}_n, t_n) \tag{4}$$

Metoda ta ma prostą interpretację geometryczną. Rozpoczynając od punktu $(t_i, \mathbf{u}_i)$ leżącego na krzywej całkowej będącej rozwiązaniem zagadnienia (1) posuwamy się po stycznej do wykresu aż osiągniemy punkt o współrzędnej niezależnej t_{i+1}.

3 Metoda Runge–Kutty

Metoda Eulera jest rzadko stosowana ze względu na swoją bardzo wolną zbieżność (musimy stosować bardzo mały krok by mieć dobrą dokładność) i niestabilność (w niektórych zagadnieniach dla dowolnie małego kroku nie uzyskamy dobrego przybliżenia). Można ją ulepszyć, biorąc styczną do krzywej nie w punkcie t_i tylko w połowie przedziału, t.j. w $t_{i+1/2}$. Wartość funkcji w połowie przedziału wyliczamy metodą Eulera z dwukrotnie mniejszym krokiem

$$\mathbf{u}_{n+1/2} = \mathbf{u}_n + \frac{\Delta t}{2}\, \mathbf{f}(\mathbf{u}_n, t_n) \tag{5a}$$
$$\mathbf{u}_{n+1} \quad = \mathbf{u}_n + \Delta t\, \mathbf{f}(\mathbf{u}_{n+1/2}, t_{n+1/2}) \tag{5b}$$

Dzięki użyciu wartości funkcji pośrodku przedziału osiągnęliśmy szybszą zbieżność metody. Skoro może to pomóc, spróbujmy zastosować kilka sposobów wyliczenia wartości funkcji pośrodku przedziału i uśrednijmy je.

$$\frac{\mathbf{u}_{n+1}-\mathbf{u}_n}{\Delta t} = \frac{1}{2}\big[\mathbf{f}(\mathbf{u}_n, t) + \mathbf{f}(\mathbf{u}_{n+1}, t)\big] \tag{6a}$$

$$\frac{\mathbf{u}_{n+1}-\mathbf{u}_n}{\Delta t} = \mathbf{f}\Big(\mathbf{u}_n + \frac{\Delta t}{2}\mathbf{f}(\mathbf{u}_n, t_n), t_n + \frac{\Delta t}{2}\Big) \tag{6b}$$

$$\frac{\mathbf{u}_{n+1}-\mathbf{u}_n}{\Delta t} = \mathbf{f}\Big(\frac{\mathbf{u}_n + \mathbf{u}_{n+1}}{2}, t_n + \frac{\Delta t}{2}\Big) \tag{6c}$$

Wprowadźmy następujące oznaczenia

$$k_1 = \mathbf{f}(\mathbf{u}_n, t_n) \tag{7a}$$

$$k_2 = \mathbf{f}(\mathbf{u}_{n+1/2}, t_{n+1/2}) \tag{7b}$$

$$k_3 = \mathbf{f}\Big(\frac{\mathbf{u}_n + \mathbf{u}_{n+1}}{2}, t_{n+1/2}\Big) \tag{7c}$$

$$k_4 = \mathbf{f}(\mathbf{u}_{n+1}, t_{n+1}) \tag{7d}$$

Aby móc efektywnie wyliczać k_i musimy wyrazić je przez wielkości już wyliczone, t.j. wyrazić k_n przez $k_i, i = 1 \ldots n-1$. Przekształcamy równanie (7c) by pozbyć się $\mathbf{u}_{n+1}$ po prawej stronie

$$\begin{aligned}\mathbf{f}\Big(\frac{\mathbf{u}_n + \mathbf{u}_{n+1}}{2}, t_n + \frac{\Delta t}{2}\Big) &= \mathbf{f}\Big(\frac{\mathbf{u}_n + \mathbf{u}_n + (\mathbf{u}_{n+1} - \mathbf{u}_n)}{2}, t_n + \frac{\Delta t}{2}\Big) = \\ &= \mathbf{f}\Big(\mathbf{u}_n + \frac{\Delta t}{2}\frac{\mathbf{u}_{n+1}-\mathbf{u}_n}{\Delta t}, t_n + \frac{\Delta t}{2}\Big) = \\ &= \mathbf{f}\Big(\mathbf{u}_n + \frac{\Delta t}{2}k_2, t_n + \frac{\Delta t}{2}\Big)\end{aligned}$$

Skorzystaliśmy tutaj z przybliżenia k_2 na pochodną.

Analogicznie pozbywamy się zależności od $\mathbf{u}_{n+1}$ po prawej stronie w definicji k_4, tym razem korzystając z przybliżenia k_3 na pochodną (a właściwie na iloraz różnicowy).

Dostajemy zatem

$$k_1 = \mathbf{f}(\mathbf{u}_n, t_n) \tag{8a}$$

$$k_2 = \mathbf{f}\Big(\mathbf{u}_n + \frac{\Delta t}{2}k_1, t_n + \frac{\Delta t}{2}\Big) \tag{8b}$$

$$k_3 = \mathbf{f}\Big(\mathbf{u}_n + \frac{\Delta t}{2}k_2, t_n + \frac{\Delta t}{2}\Big) \tag{8c}$$

$$k_4 = \mathbf{f}(\mathbf{u}_n + \Delta t\, k_3, t_n + \Delta t) \tag{8d}$$

Aby wyliczyć dobre przybliżenie na pochodną (na iloraz różnicowy) uśredniamy wzory (6a ... 6c).

$$\frac{\mathbf{u}_{n+1}-\mathbf{u}_n}{\Delta t}=\frac{1}{3}\big((6a)+(6b)+(6c)\big)=\frac{1}{6}\big(k_1+2k_2+2k_3+k_4\big) \qquad (9)$$

Ostatecznie dostajemy następującą metodę (klasyczną metodę Runge–Kutty czwartego rzędu)

$$\mathbf{u}_{n+1}=\mathbf{u}_n+\frac{\Delta t}{6}(k_1+2k_2+2k_3+k_4) \qquad (10)$$

Warto zwrócić uwagę, że w metodzie Runge–Kutty musimy wielokrotnie obliczać wartość funkcji. Jeśli zatem obliczanie wartości funkcji zajmuje dużo czasu, to lepiej stosować inne metody.

4 Implementacja klasycznej metody Runge–Kutty czwartego rzędu

Jedna z implementacji klasycznej metody Runge–Kutty czwartego rzędu. Rozwiązuje układ równań różniczkowych zwyczajnych zadanych za pomocą wektora funkcyj prawych stron.

```
typedef double real;
#define REAL "%lg"
#define _R "l"

#define ZERO 2.0E-150

#define sqr(x) ((x)*(x))

#define FALSE 0
#define TRUE 1

#define max(x,y) ((x)>(y)?(x):(y))
#define min(x,y) ((x)<(y)?(x):(y))

/* czesc prawej strona rownania rozniczkowego zwyczajnego */
/* pierwszego rzedu                                       */
/*                                                        */
/*   x1' = f1( t, x1, x2..., xN )                         */
/*   x2' = f2( t, x1, x2..., xN )                         */
/*   ...                                                  */
```

```
/*   xN'   = fN( t, x1, x2 ..., xN )                                */
/*                                                                  */
typedef real (*p_dequation1)( real t, real* x);

void RungeKutta4_classic(
  real *res,          /* wynik: u_n+1  */
  int dim,            /* wymiar ukladu */
  p_dequation1 *f,    /* prawa strona  */
  real* x,            /* punkt u_n     */
  real t,
  real h )            /* krok: Dt      */
{
  int i;
  real *k1,*k2,*k3,*k4;
  real *tmp;

  /* UWAGA: powinno sie sprawdzac, czy alokacja pamieci */
  /*          sie udala czy nie                         */
  k1  = (real *)calloc(dim,sizeof(real));
  k2  = (real *)calloc(dim,sizeof(real));
  k3  = (real *)calloc(dim,sizeof(real));
  k4  = (real *)calloc(dim,sizeof(real));
  tmp = (real *)calloc(dim,sizeof(real));

  for(i = 0;i < dim; i++)
    k1[i] = (*f[i])(t,x);
  for(i = 0;i < dim; i++)
    tmp[i] = x[i] + 0.5*h*k1[i];
  for(i = 0;i < dim; i++)
    k2[i] = (*f[i])(t + 0.5*h,tmp);
  for(i = 0;i < dim; i++)
    tmp[i] = x[i] + 0.5*h*k2[i];
  for(i = 0;i < dim; i++)
    k3[i] = (*f[i])(t + 0.5*h,tmp);
  for(i = 0;i < dim; i++)
    tmp[i] =  x[i] + h*k3[i];
  for(i = 0;i < dim; i++)
    k4[i] = (*f[i])(t + h    ,tmp);

  for(i = 0;i < dim; i++)
    res[i] = x[i] + h*(k1[i]+2*k2[i]+2*k3[i]+k4[i])/6.0;

  free(tmp);
```

```
  free(k4);
  free(k3);
  free(k2);
  free(k1);
}
```

Implementacja metody Runge–Kutty z „Numerical Receipies in C”

```
#define NRANSI
#include "nrutil.h"

void rk4(float y[], float dydx[], int n, float x, float h, float yout[],
        void (*derivs)(float, float [], float []))
/* Majac dane wartosci zmiennych y[1..n] i ich pochodnych dxdy[1..n]   */
/* w punkcie x, stosuje metode Runge-Kutty do obliczenia rozwiazania   */
/* w punkcie x+h. Zwraca obliczone rozwiazanie w yout[1..n], ktory nie */
/* musi byc rozlaczny z y (t.j. moze byc to ta sama tablica).          */
/* Uzytkownik powinien dostarczyc funkcje derivs(x,y,dxdy), ktora      */
/* zwraca pochodne dxdy w punkcie x                                    */
{
  int i;
  float xh,hh,h6,*dym,*dyt,*yt;

  /* alokacja pamieci */
  dym=vector(1,n);
  dyt=vector(1,n);
  yt=vector(1,n);

  hh=h*0.5;
  h6=h/6.0;
  xh=x+hh;

  /* wyliczenie k_1,...,k_4 */
  for (i=1;i<=n;i++) yt[i]=y[i]+hh*dydx[i];
    (*derivs)(xh,yt,dyt);
  for (i=1;i<=n;i++) yt[i]=y[i]+hh*dyt[i];
    (*derivs)(xh,yt,dym);
  for (i=1;i<=n;i++) {
    yt[i]=y[i]+h*dym[i];
    dym[i] += dyt[i];
  }
  (*derivs)(x+h,yt,dyt);

  /* wyliczenie rozwiazania */
```

```
    for (i=1;i<=n;i++)
        yout[i]=y[i]+h6*(dydx[i]+dyt[i]+2.0*dym[i]);

    /* zwolnienie pamieci */
    free_vector(yt,1,n);
    free_vector(dyt,1,n);
    free_vector(dym,1,n);
}
#undef NRANSI
/* (C) Copr. 1986-92 Numerical Recipes Software ]*!. */
```