COTOROBAI VLAD DOGARU IASMINA

RUNGE-KUTTA 4 METHOD

1. **Introducere a problemei rezolvate (1-2-3 paragrafe, cu citare de unde ati luat algoritmul)**

Problemele care implică ecuații diferențiale obișnuite (ODE) pot fi întotdeauna reduse la studiul seturilor de ecuații diferențiale de ordin întâi. Un tip major de metode numerice practice pentru rezolvarea problemelor de valoare inițială pentru ODE sunt Metodele Runge-Kutta. [2]

Metodele Runge-Kutta propagă o soluție pe un interval prin combinarea informațiilor din mai mulți pași în stil Euler, și apoi folosind informațiile obținute pentru a se potrivi cu o expansiune a seriei Taylor până la un ordin mai mare.

Runge-Kutta este ceea ce utilizați atunci când:
(i) nu știți mai bune metode,
(ii) aveți o problemă intransigentă în care Bulirsch-Stoer eșuează
(iii) aveți o problemă banală în care eficiența computațională nu este îngrijorătoare.

Runge-Kutta reușește practic întotdeauna; dar nu este de obicei cea mai rapidă.

În acest proiect am ales pentru prezentare metoda Runge-Kutta de ordin 4. Spre deosebire de metoda lui Euler, care calculează o pantă la un interval, Runge-Kutta 4 calculează patru pante diferite și le folosește ca medii ponderate. Aceste pante sunt denumite în mod obișnuit k1, k2, k3 și k4, iar programul trebuie să le calculeze la fiecare pas.

2. **Implementare**

-> Subrutinele implementate (de ce? ce fac? parametrii transmisi)

Avem 2 subrutine implementate:

1) Subrutina f:
   a. A fost implementată, deoarece la fiecare iterație a for loop-ului, parametrii funcției f își modifică valoarea.
   b. f este funcția derivată, care calculează f(var_a, var_b) și returnează rezultatul în registrul $f3 din Coproc1.
   c. Parametrii transmiși: var_a și var_b..

2) Subrutina display_x0_y0_yn:
   a. A fost implementată, deoarece la fiecare iterație a for loop-ului, se afișează valorile x0, y0, yn curente.
   b. Afișează valorile variabilelor x0, y0 și yn.
   c. Parametrii transmiși: x0, y0 și yn.

-> Blocking points -> cum ati implementat, cum ati rezolvat problema?

Am ales algoritmul din varianta din sursa [2], din motiv că ne-a părut algoritmul prezentat în [1] mai dificil.

-> Decizii de implementare

Implementarea algoritmului a fost realizat în programul MARS, un IDE pentru programare în limbaj de asamblare MIPS.

3. **Exemplu de functionare ->**

Rezultatul acestui program este soluția pentru **(x – y) / 2** cu condiția inițială **y = 1** pentru **x = 0** adică **y(0) = 1** și încercăm să evaluăm această ecuație diferențială la **x = 0.4** în două etape, adică **n = 2**. (Aici **x = 0.4,** adică **y(0.4) = ?** este punctul nostru de calcul)

RK4.c

C program for RK4

RK4.asm

MIPS program for RK4

Input values:

```
Enter Initial Condition
x0 = 0
y0 = 1
Enter calculation point xn = 0.4
Enter number of steps: n = 2
```

```
Enter initial condition
x0 = 0
y0 = 1
Enter calculation point
xn = 0.4
Enter number of steps
n = 2
```

Output values:

```
x0      y0      yn
0.0000  1.0000  0.9145
0.2000  0.9145  0.8562

Value of y at x = 0.40 is 0.856
```

```
x0      y0      yn
0.0     1.0     0.9145125
0.2     0.9145125       0.8561927
The result is:
xn = 0.4
yn = 0.8561927
```

Discutii privind rezultate

h = (xn - x0)/n = (0.4 – 0) / 2 = 0.2

k1 = h * (f(x0, y0)) = 0.2 * f(0, 1) = 0.2 * (0 – 1) / 2 = -0.1

k2 = h * (f((x0 + h/2), (y0 + k1/2))) = 0.2 * f(0.1, 0.95) = 0.2 * (0.1 – 0.95) / 2 = -0.085

k3 = h * (f((x0 + h/2), (y0 + k2/2))) = 0.2 * f(0.1, 0.9575) = 0.2 * (0.1 – 0.9575) / 2 = -0.08575

k4 = h * (f((x0 + h), (y0 + k3))) = 0.2 * f(0.2, 0.91425) = 0.2 * (0.2 – 0.91425) / 2 = -0.071425

k = (k1+ 2*k2 + 2*k3 + k4)/6 = -0.0854875

yn = y0 + k = 1 - 0.0854875 = 0.9145125 **<- rezultatul teoretic dupa primul pas**

Dupa imaginele de mai sus, putem constata că atât în C, cât și în MIPS rezultatele sunt identice.

De asemenea , rezultatul teoretic din primul pas coincide cu cel din primul pas din cele două program, respectiv valorile returnate sunt conform celor teoretice si sunt valide

4. Concluzii

4.1 -> Blocking points

Modalitatea sugerată în [2], necesita utilizarea pointerilor la funcție și alocarea vectorilor, iar metoda aleasă necesită parcurgerea unui singur for loop, unde se afișează pe parcurs rezultatele intermediare.

4.2 -> Comentarii legate de implementare

Ne-am folosit de variabilele din memorie, declarate la începutul segmentului .data, de regiștrii coprocesorului 1 și regiștrii pentru numerele întregi, pentru a actualiza fiecare pas al algoritmului în loop.

4.3 -> What I learned?

Am învățat o metodă nouă de soluționare a problemelor ce implica ecuații diferențiale obișnuite. De asemenea, am învățat să implementăm în limbaj de asamblare algoritmul Runge-Kutta, folosindu-ne de arhitectura MIPS.

5. Bibliografie

[1] Ordinary Differential Equation Using Fourth Order Runge Kutta (RK) Method Using C (codesansar.com)
[2] NumericalRecipesinC.pdf (grad.hr)
[3] MARS MIPS simulator - Missouri State University