NINDE 2016 / 17, 2. domača naloga

Jure Slak, 27152005

Naloga 1.
Najprej je implementirana splošna eksplicitna RK metoda za reševanje sistemov NDE v datoteki `RKEplicit.m` s pomočjo nekaj pomožnih funkcij. V datoteki `make_RK4.m` je skripta, ki naredi shemo za RK4 in naredi objekt, ki predstavjla solver po tej metodi. Podobno so v skripti `make_CK.m` podatki za obe RK metodi, na katerih bazira Cash-Karp metoda (ter tudi podatki za shemo metode, predstavljene kot Algoritem 1.). Funkcija `CashKarp` je implementirana kot piše na nalogi. Če uporabimo algoritem 1, potem je pri natančnosti 10^{-6} $h$konstantno enak $hmin$, kar je tudi logično, saj ocena napake ni pravilna, in ne kaže dejanske napake. Če pa uporabimo dejansko Cash-Karpovo metodo, pa se h res prilagaja in tudi opravi veliko manj izračunov. Celotna rešitev naloge je v `nal1.m`. Prava Cash-Carp metoda, kot je napisana v algoritmu 1, nikoli ne doseže vrednosti $x = 2$, saj je do tja premajhen korak. Ker ni specificirano, kaj naj naredimo v tem primeru, sem jaz vseeno naredil majhen korak in izračunal vrednost $y(2)$.

```
CK-prava: y(2) = 1.9506227004274028
CK-alg1:  y(2) = 1.9684232501488073
RK4:      y(2) = 1.9506785300973337
```

Pri implementaciji metod bi si lahko shranjevali že izračunane vrednosti f in s tem pospešili izvajanje.

Naloga 2.

Implementacija obeh metod je direktna, ena v datoteki `AdamsBashSistem.m` in druga v `MilneSistem.m`. Naloga je rešena v datoteki `nal2.m`.

```
h = 0.1
Approx for y(10) using AB: -0.3498705377962865
Approx for y(10) using MI: -0.4414240151890469
h = 0.05
Approx for y(10) using AB: -0.4227331662395586
Approx for y(10) using MI: -0.4222779703683874
```

Naloga 3.
Najprej rešimo nalogo z RK4 in iz slike preberemo približek za peto ničlo. Nato s standardno tangentno metodo iščemo ničlo, pri čemer odvode in vrednosti aproksimiramo z RK4, ki se od prejšnje ničle do naslednje sprehodi z maksimalnim korakom h, ali pa z manjšim, če je celotna razdalja manjša. Metoda se lahko sprehaja v levo ali v desno. Seveda pa je ničla bolj od tega kakšno natančnost si zberemo pri Newtonovi iteraciji odvisna od začetnega h. Newtonova iteracija pričakovano skonvergira v približno petih korakih tudi ob toleranci 10^{-16}. Za začetni približek sem vzel 7.5.

```
h = 0.1
korak: 1, priblizek: 7.811265563327425
korak: 2, priblizek: 7.757021945794063
korak: 3, priblizek: 7.757430943970567
korak: 4, priblizek: 7.757430943793660
korak: 5, priblizek: 7.757430943793660
Nicla: 7.7574309437936604.

h = 0.01
korak: 1, priblizek: 7.811097555023491
korak: 2, priblizek: 7.756912740566238
korak: 3, priblizek: 7.757320650664874
korak: 4, priblizek: 7.757320650489378
korak: 5, priblizek: 7.757320650489378
Nicla: 7.7573206504893779.
```