Modelleren, Simuleren & Continüe Wiskunde
Opdracht 3: Euler, Runge-Kutta, ODE's

Taddeüs Kroes (6054129) Sander van Veen (6167969)

10 april 2011

1 De opdracht

De opdracht bestaat uit twee delen: het eerste deel betreft het implementeren van de Euler- en Runge-Kutta-methoden, in het tweede deel worden deze methoden vergeleken en gebruikt om enkele ODE's te benaderen. De opdrahchten zelf zullen we niet heel uitgebreid bespreken, alleen de antwoorden die we erop hebben.

2 Implementatie

We hebben de drie methoden volgens de gegeven functievoorschriften geïmplementeerd in C (zie het bestand *integration.c*). We hebben daarbij een "logger" gemaakt die een rij waardes naar `stdout` print. We gebruiken dit als invoer voor het python-script *plot.py*, dat deze waarden in een grafiek zet. Een voorbeelduitvoer is:

```
$ ./main 5 | ./plot.py gilpin.svg
```

Dit voert de integratie van ODE nr. 5 (Gilpin's model) uit en plot de resultaten in een grafiek welke wordt opgeslagen als de afbeelding *gilpin.svg*.

De implementatie van de integratiemethoden zelf is gedaan met behulp van de pseudocode in het boek. Deze is volgens ons vrij duidelijk en daarom zullen we er hier niet dieper op ingaan.

3 Testfuncties

De testfuncties kunnen worden uitgevoerd met het programma `./test` (bestand *test.c*). Dit bestand schrijft de log data naar `/dev/null` waardoor alleen de "custom" output wordt geprint. De data op zich vinden we namelijk weinig interessant, we willen vooral weten wat de waarden op `t1` zijn.

Van vier van de vijf functies berekenen we de beoogde waarden exact en vergelijken deze met de waarden die het programma berekent, dit kan omdat de functies

redelijk eenvoudig zijn. Alle methoden benaderen de beoogde waarden steeds beter naarmate we `DT` kleiner kiezen, dus concluderen we dat de methoden correct zijn geïmplementeerd.

Een apart geval is de vierde testfunctie (*"f = y * y, with t0 starting at -1 and y0 at 1, integrating to t = 1 (this should not work, why?)"*). Deze integratie lever voor alle methoden `INFINITY` op, dit wordt veroorzaakt door een verkeerd gekozen startwaarde y0.

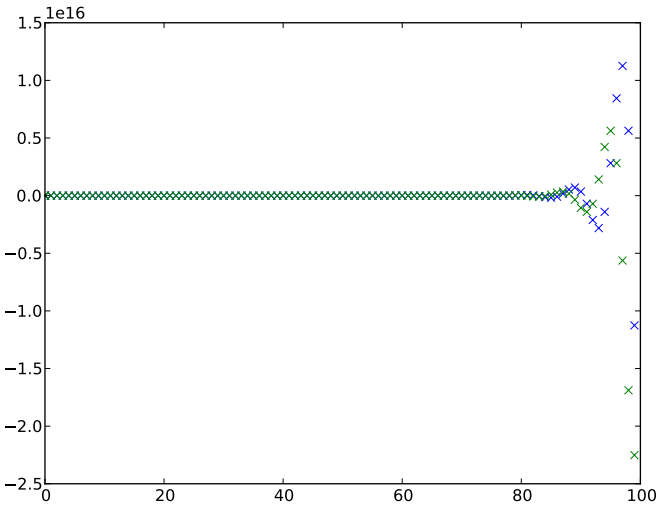
We zien dat, vooral bij een grote `DT`, dat Euler verschilt van RK2 en RK4. RK2 ligt redelijk dicht bij RK4, ook bij niet al te kleine waarden van `DT`. RK4 ligt het dichtst bij de exacte waardes, we kunnen dus concluderen dat deze het meest nauwkeurig is (niet verassend, want RK4 voert de meeste operaties uit).

4 ODE's

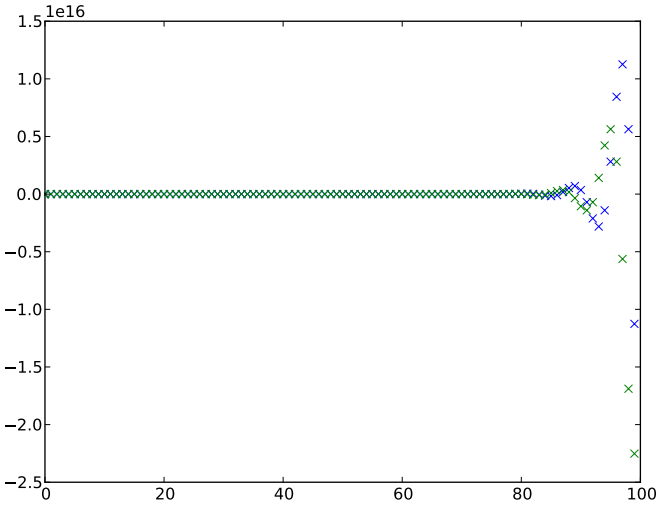
4.1 Harmonische oscillator

De vergelijking van de drie integratiemethoden kan worden uitgevoerd met:

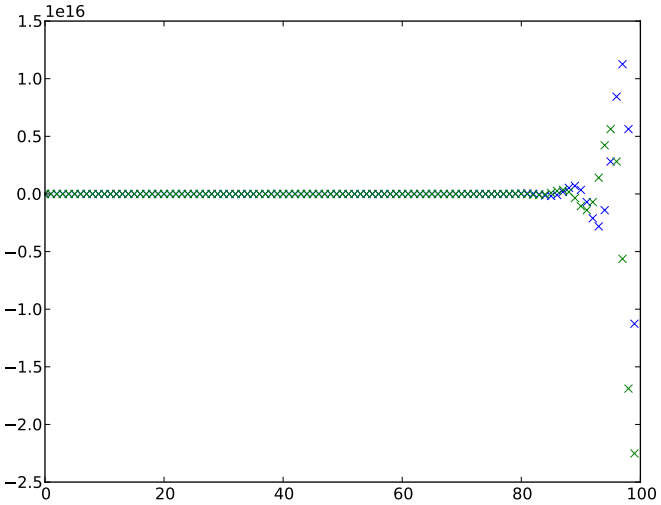
```
$ sh ./compare_osc.sh
```



Figuur 1: Euler



Figuur 2: Runge-Kutta 2



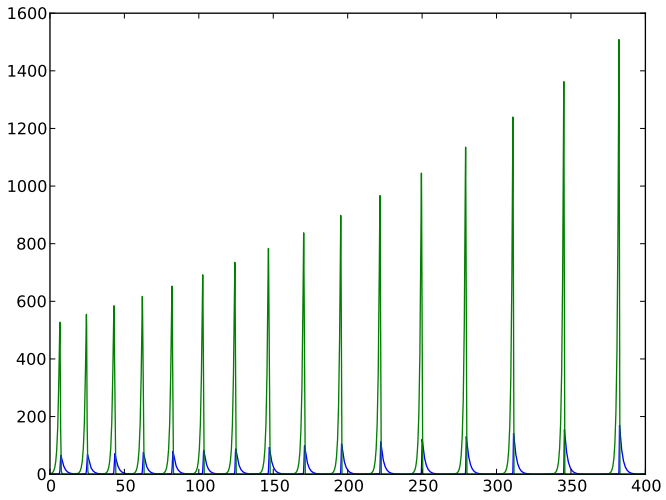
Figuur 3: Runge-Kutta 4

4.2 Damped, forced harmonic oscillator

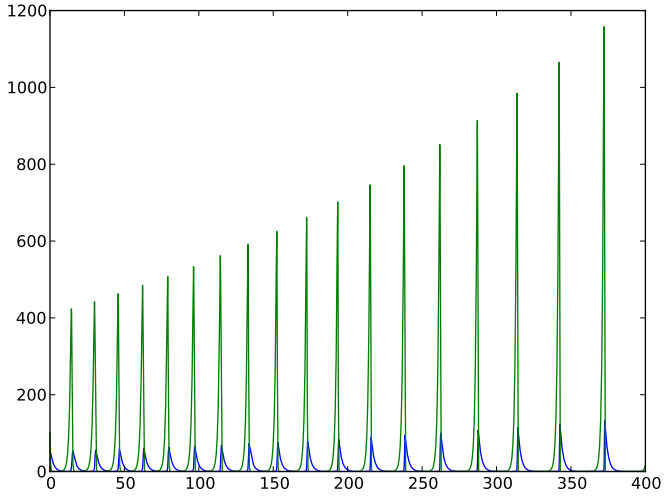
4.3 Lottka-Voltera

We hebben de Lotka-Volterra simulatie met de opgegeven constanten uitgevoerd voor verschillende startwaarden. De ene keer met startwaarden op het “stabiele

punt" en de andere keer met startwaarden daar ver vandaan. Het stabiele punt ligt op $y = \frac{a}{b} = 0.5$ en $x = \frac{c}{d} = 1.0$.



Figuur 4: $x(0) = 0.5$, $y(0) = 1.0$

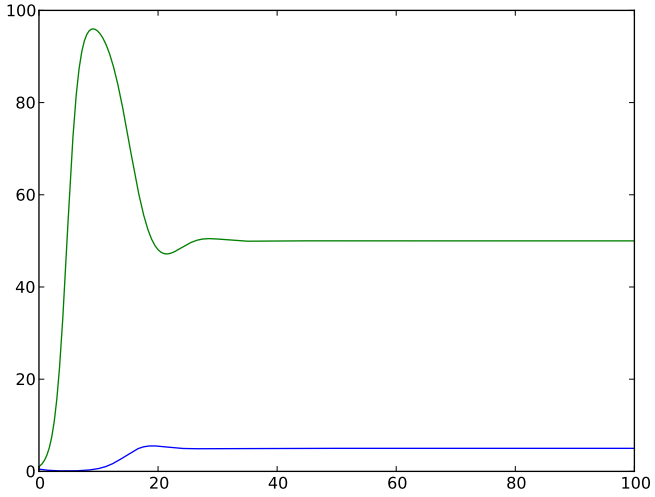


Figuur 5: $x(0) = 100$, $y(0) = 200$

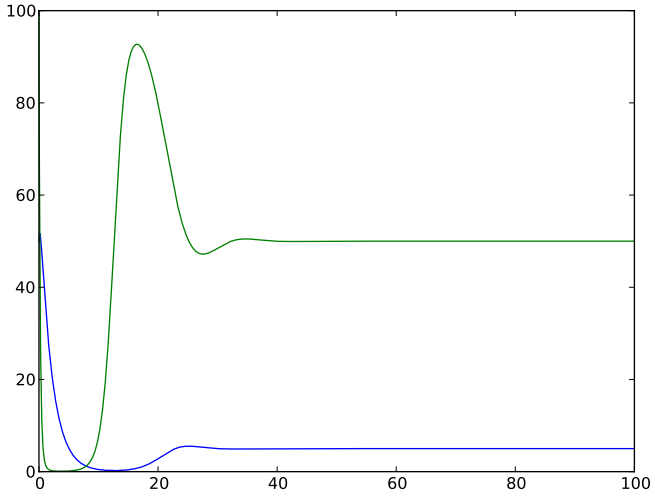
We zien dat de grafiek bij de hoge waardes een kleiner periode heeft, dus vaker fluctueert. Bij het stabiele punt zijn er grotere periodes, waardoor de prooi sneller een groot aantal krijgt.

4.4 Modified predator-prey model

We gebruiken exact dezelfde startwaarden als bij Lotka-Volterra en nu krijgen het volgende resultaat:



Figuur 6: $x(0) = 0.5$, $y(0) = 1.0$



Figuur 7: $x(0) = 100$, $y(0) = 200$

We zien dat in beide gevallen de grafiek niet boven de 100 uitkomt (namelijk omdat $y_m = 100$, dit is de maximumwaarde). De tweede grafiek vertoont na

verloop van tijd min of meer hetzelfde gedrag als de eerste, alleen wordt in het eerste gedeelte de stabiele positie “opgezocht” waarna naar dezelfde evenwichtsstand wordt bewogen.

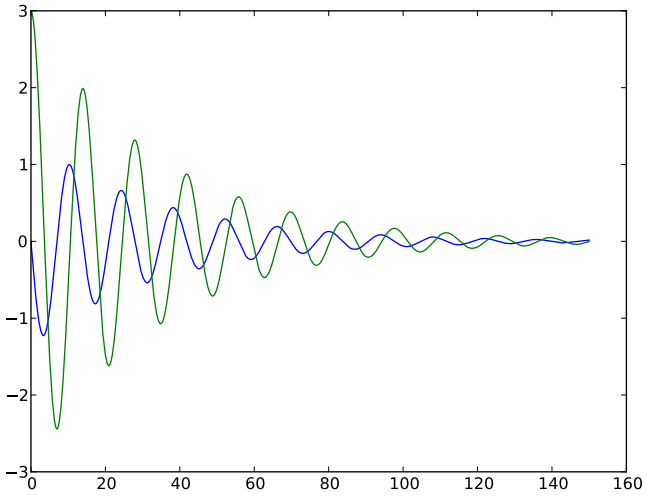
Hieruit concluderen we dat de startwaarden waarschijnlijk weinig uitmaken, omdat na verloop van de tijd de grafiek toch wel een evenwichtsstand bereikt.

4.5 PID controller

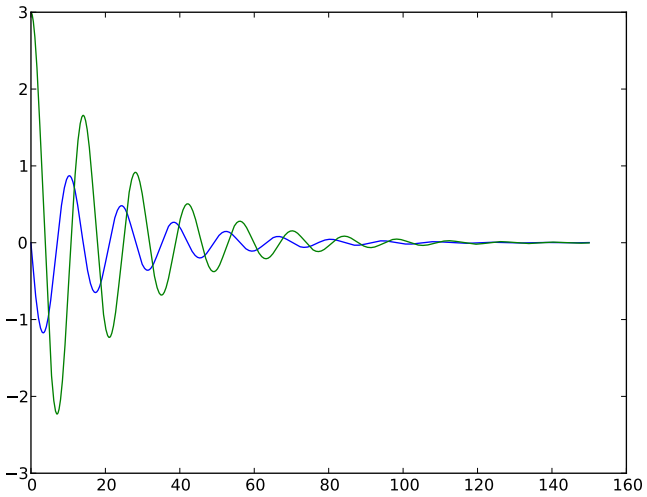
De constante P is de mate waarin de afstand tot de doelpositie (s) meetelt voor de snelheid van de beweging. Als $s(0)$ positief gekozen wordt moet P negatief zijn (bijv. $3 > 0$, als $s(0) = 3$, er moet naar $s = 0$ worden bewogen dus een negatieve beweging), en andersom.

D is de mate waarin de bewegingssnelheid wordt beperkt, men wil waarschijnlijk niet dat de arm een bepaalde snelheid overschrijdt. Omdat D een beperkende factor is, is deze ook negatief.

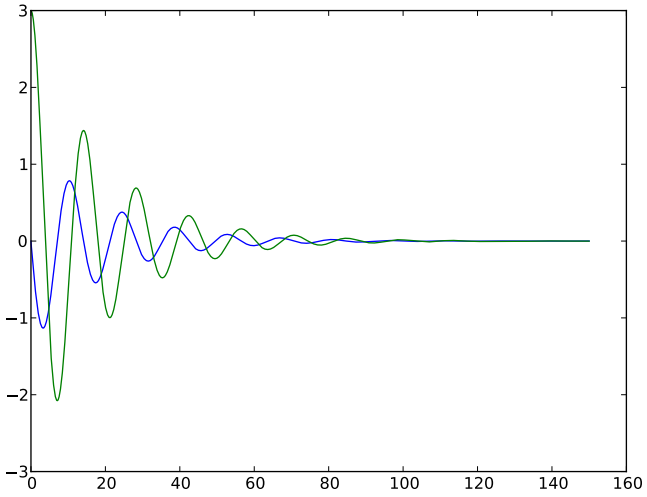
d is de delay voor het bewegen voor de arm, we hebben voor verschillende waarden hiervan de grafiek geplot. We hebben hierbij gekozen voor $P = -3$, $D = -1$, $a = 0.1$ en $m = 10$ als constanten.



Figuur 8: $d = 1.0$



Figuur 9: $d = 0.5$



Figuur 10: $d = 0.1$

We zien dat hoe kleiner d, hoe vlugger de grafiek een evenwichtsstand bereikt. Dit is logisch, want de arm reageert sneller als de delay kleiner is.

4.6 Gilpin's Model