OPDRACHTEN

PROGRAMMEREN

(5e kwartaal natuur- en sterrenkunde)

August 31, 2009

Faculteit der Natuurwetenschappen, Wiskunde en Informatica
P.F. Klok
J.M. Thijssen
W.L. Meerts

Radboud Universiteit Nijmegen

Opdrachten A

A.1 Inleiding

Het doel van Programmeren is om je bekend te maken met de computer en om je, aan de hand van vooral fysische voorbeelden, te leren op een algoritmische manier problemen op te lossen. In deze opdrachten gaat het erom zulke algoritmen in de programmeertaal C te verwezenlijken.
Naast fysische problemen komen ook algemene zaken aan de orde die belangrijk zijn in de informatica, zoals het werken met teksten en sorteren van gegevens.

In de moderne fysica speelt de computer een steeds grotere rol: in vrijwel alle gebieden van de fysica wordt de computer toegepast en dit gebeurt op een zeer veelzijdige manier.
Men kan de toepassingen als volgt indelen:

- **Numerieke analyse:** het getalmatig oplossen van vergelijkingen waar analytische methoden tekort schieten of onwerkbaar ingewikkeld zijn.
- **Besturing en gegevensverwerking** bij experimentele opstellingen.
- **Simulaties** van modellen om experimenten te verklaren of om de modellen zelf beter te begrijpen.
- **Grafische toepassingen:** visualiseren van processen of van numeriek bepaalde oplossingen van vergelijkingen.
- **Algebraïsche manipulatie:** *symbolisch,* dus niet getalmatig, verwerken van vergelijkingen.

In de opdrachten zullen, naast niet specifiek fysische problemen, de eerste vier toepassingsgebieden aan de orde komen.

De opdrachten worden alleen gemaakt. Niet alle opdrachten dienen ingeleverd te worden. Op het college wordt een lijst verstrekt met in te leveren opdrachten. Alleen de in te leveren opdrachten tellen mee in de eindbeoordeling.

A.2 Opdrachten

1. **Verwisselen**
   Schrijf een programma dat twee **float** variabelen, **a** en **b**, vult met twee vanaf het toetsenbord in te lezen getallen en dat vervolgens de inhoud van **a** en **b** verwisselt en op het scherm afdrukt:

   (a) m.b.v. invoering van een hulpvariabele **c**,

   (b) zonder invoering van een hulpvariabele.

2. **Meetkunde**
   Schrijf een programma dat vraagt om een hoek (in *graden*) en de lengte van de twee aanliggende zijden van een driehoek en dat vervolgens de lengte geeft van de overstaande zijde en de cosinus van de twee overige hoeken.
   Gebruik hierbij o.a. de standaardfunctie **cos**:

   cos (x)

   die de cosinus van **x** levert, waarbij **x** in *radialen* gegeven is.

3. **Rekenen**
   Schrijf een programma dat de gebruiker vraagt een reëel getal **x** op te geven en dat vervolgens de waarde van

   $$\frac{\log|1+2x|}{(1-2x)}$$

   afdrukt op het scherm.
   Maak gebruik van de standaardfunctie

   log (x)

   die voor positieve **x** van type **float** de natuurlijke logaritme oplevert.
   Test het programma in ieder geval voor de waarden **x=-0.5**, **x=0.5** en **x=-1**.

4. **Vierkantsvergelijking**
   Schrijf een programma dat aan de gebruiker vraagt om de coëfficienten **a**, **b** en **c** op te geven van de kwadratische vergelijking voor de variabele **x**:

   $$ax^2 + bx + c = 0$$

   en vervolgens de oplossingen van deze vergelijking op het scherm afdrukt.
   Houd rekening met complexe oplossingen en met het nul zijn van **a**!
   Gebruik de standaard functie **sqrt**:

   ```
   sqrt (x)
   ```

   die $\sqrt{x}$ levert als $x \geq 0$.

5. **Kalender**
   Schrijf een programma dat aan de gebruiker vraagt om een datum (dag, maand en jaar) op te geven en dat vervolgens op het scherm schrijft welke dag van de week (maandag, ... zondag) bij die datum hoort. Zorg dat het programma in ieder geval voor de 20ste en de 21ste eeuw werkt.
   NB: **Schrikkeljaren** zijn de jaren die deelbaar zijn door 4, **uitgezonderd** de eeuwjaren die niet deelbaar zijn door 400.

6. **Botsing**
   Beschouw een botsing met botsingsparameter b, tussen twee harde bollen, 1 en 2, met straal a_1 en a_2, en massa m_1 en m_2. Bol 2 ligt stil en 1 nadert 2 met snelheid v_1.
   Schrijf een programma dat vraagt om a_1, a_2, m_1, m_2, b en v_1 en dat vervolgens de richting en grootte van de eindsnelheden van 1 en 2 geeft.

   AANWIJZING.
   Beschouw in het CM systeem een puntdeeltje dat tegen een bol met straal $a_1 + a_2$ geschoten wordt. De impuls van dit deeltje is de *relatieve* impuls $\mathbf{p}_1 - \mathbf{p}_2$ van de twee bollen (in het CM systeem).

Intermezzo

In de hierna volgende opgaven is het soms mogelijk, of vereist, om grafisch weer te geven wat er berekend wordt. Hiertoe kun je gebruik maken van de grafische bibliotheek OpenGL. Deze bibliotheek bevat een aantal functies die het grafische leven eenvoudiger maken. Een belangrijk voordeel is bijvoorbeeld dat je alle tekeningen in gebruikerscordinaten kunt maken.
Links naar beschrijvingen, voorbeelden en toepassingen van OpenGL vind je op web site van dit college: `http://www.mlf.science.ru.nl/leo_meerts/programmeren/`.

7. Lineaire fit

De kleinste-kwadratenmethode wordt gebruikt om een goed passende rechte lijn door een aantal meetpunten te tekenen.
Beschouw een rij van N meetpunten (X_i, Y_i) met $X_i < X_j$ voor $i < j$.
Voor de beste fit $y = mx + b$ door de meetpunten geldt dan dat:

$$m = \frac{N\sum_i X_i Y_i - \sum_i X_i \sum_j Y_j}{N\sum_i X_i^2 - (\sum_i X_i)^2}$$

$$b = \frac{(\sum_i {X_i}^2)\sum_j Y_j - \sum_i X_i \sum_j X_j Y_j}{N\sum_i X_i^2 - (\sum_i X_i)^2}$$

Schrijf nu een programma dat een rij waarden (X_i, Y_i) inleest en deze in een grafiek tekent.
Teken in deze grafiek ook de rechte lijn die berekend is volgens bovenstaande formules. Maak bij deze opgave gebruik van OpenGL.

8. Interpolatie

We willen het nulpunt x_0 van een functie f gaan bepalen. Het nulpunt wordt gevonden met een bepaalde nauwkeurigheid: dit kan de nauwkeurigheid zijn waarbinnen f van nul mag afwijken, òf waarmee de gevonden waarde van x van x_0 mag afwijken.

(I) (a). Beschouw eerst (strikt) stijgende functies op het interval $[0, 1]$.
Bedenk een methode waarmee het mogelijk is om met n stappen x_0 met een nauwkeurigheid van 2^{-n} te bepalen. Schrijf een procedure die dit algoritme uitvoert.
Test deze procedure voor $f(x) = x^2 + 3x - 1$.

(b). Idem, voor functies die òf strikt stijgend òf strikt dalend zijn op $[0, 1]$.
Test deze procedure voor $f(x) = -x^2 - 3x + 1$.

(c). Idem, voor functies die meerdere nulpunten hebben op $[0, 1]$, die onderlinge afstand $> \Delta$ hebben (neem aan dat er geen dubbele nulpunten zijn).
Test deze procedure voor $f(x) = (x - \frac{1}{4})(x - \frac{1}{2})(x - \frac{3}{4})$. Bij deze procedure moet dus een schatting van Δ gemaakt worden door de gebruiker, aangezien de procedure zelf natuurlijk niet kan weten wat de minimale afstand tussen de nulpunten is.

(d). Bedenk een snellere versie van deze procedure, die op hetzelfde idee gebaseerd is (je hoeft dit niet te programmeren).

(II) We gaan weer het nulpunt bepalen van een functie f. Dit keer gebruiken we de zg. *Newton-Raphson methode*. Hierbij wordt aangenomen dat naast de functie f ook de afgeleide f' bekend is op, in ons geval $[0, 1]$. De Newton-Raphson methode werkt als volgt. Er wordt een punt x genomen in de buurt van het nulpunt x_0 (in ons geval nemen we $x = 0$). Aangenomen wordt, dat $f' > 0$ op $[0, 1]$. Er wordt nu een schatting gemaakt van het nulpunt door het snijpunt van de X-as met de raaklijn aan f in het probeerpunt x te bepalen. Daarna nemen we dit nieuwe punt als schatting van het nulpunt en herhalen dezelfde stap, nu voor dit nieuwe punt.

(a). Programmeer deze methode voor dezelfde gevallen als in (I).

(b). Bedenk een methode om de snelheden van de procedures uit het vorige onderdeel met de Newton-Raphson procedure te vergelijken en voer deze snelheidstest uit.

(c). Probeer een afschatting te geven voor de convergentiesnelheid van de Newton-Raphson methode in vergelijking met de bisectiemethode uit (I).

9. Matrices

(I) Schrijf een programma dat twee matrices inleest vanuit twee aparte bestanden, de matrices met elkaar vermenigvuldigt als dit mogelijk is (rij maal kolom) en het resultaat afdrukt op het scherm èn wegschrijft in een bestand. Je hebt hier twee opties voor het inlezen van de matrices vanuit de bestanden:

(a). Dit is een minimale, snelle en meest simpele oplossing. Hierbij ga je ervan uit dat op de eerste regel van het input bestand het aantal rijen (N_{rij}) en kolommen (N_{kolom}) staat in een vast formaat. En vervolgens dat er N_{rij} regels volgen met op elke regel de N_{kolom} kolom waardes van de gegeven rij in een vast formaat, bijvoorbeeld de elementen gescheiden met **één** spatie. Een goed werkende versie levert maximaal 7 punten op.

(b). Meer uitdagend is de volgende optie. In dit geval is bij het openen van het bestand niet bekend hoeveel rijen en kolommen de matrix heeft. We weten alleen dat dat op elke regel een rij staat. De getallen kunnen gescheiden zijn door spatie(s) (meerdere achter elkaar mogelijk), een "," of een TAB; en wel door elkaar heen gebruikt! Begin en eind van de regel mag ook een of meerdere spaties bevatten. Wel moeten er op iedere regel natuurlijk evenveel kolom-waardes staan (check dit!). Deze optie geeft de gebruiker optimale vrijheid in het maken van zijn input bestand, maar eist uiteraard meer van de programeur. Kies je hiervoor dan kan dat 10 punten opleveren.

Hints:

(a). Gebruik de functie *fscanf* voor het geformateerd inlezen van de eerste regel van optie (a). Lees de volgende regels in met *fgets*. Zie ook volgende hint b-B.

(b). Als je optie (b) kiest,

A. Om het aantal regels (rijen van de matrix) te tellen moet je het bestand twee keer inlezen met daartussen een *rewind*.

B. Gebruik de functie *fgets* voor het inlezen van een hele regel. Houdt er rekening mee dat er nog een LineFeed (Linux) en mogelijkerwijs **ook** een CarriageReturn (DOS) in de string kan staan en verwijder die eerst, zodat je een nette C-string hebt.

C. Vervang "," en TAB's door spaties

D. Verwijder begin en eind spaties en multiple spaties

E. *Knip* vervolgens de string in de stukjes waar de waarde van een enkel matrix element in staat en converteer elk stukje string naar een getal met behulp van de functie *strtod*.

(II) We proberen nu het programma van (I) sneller te laten lopen door het aantal tijdrovende vermenigvuldigingen te reduceren. Realiseer je hiertoe, dat als we twee vectoren $(x_1, \cdots, x_n)$ en $(y_1, \cdots, y_n)$ willen vermenigvuldigen (inprodukt), we dit rechttoe-rechtaan kunnen doen:

$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + \cdots + x_n y_n \qquad (a)$$

of op de volgende wijze:

$$\left.\begin{array}{ll} \mathbf{x} \cdot \mathbf{y} = & \sum_{i=1}^{n/2} (x_{2i-1} + y_{2i})(x_{2i} + y_{2i-1}) - \\ & \sum_{i=1}^{n/2} x_{2i-1} x_{2i} - \sum_{i=1}^{n/2} y_{2i-1} y_{2i} \end{array}\right\} n \text{ even}$$

$$\left.\begin{array}{ll} \mathbf{x} \cdot \mathbf{y} = & \sum_{i=1}^{(n-1)/2} (x_{2i-1} + y_{2i})(x_{2i} + y_{2i-1}) - \\ & \sum_{i=1}^{(n-1)/2} x_{2i-1} x_{2i} - \sum_{i=1}^{(n-1)/2} y_{2i-1} y_{2i} + \\ & x_n y_n. \end{array}\right\} n \text{ oneven} \qquad (b)$$

Formule (a) kost n vermenigvuldigingen en (b) ca. $\frac{3}{2}n$. De tweede en derde term in de tweede formule bevatten echter alleen maar producten van de componenten van $\mathbf{x}$ zelf, resp. van $\mathbf{y}$ zelf. Aangezien we de rijen en kolommen van de matrices A en B die met elkaar vermenigvuldigd worden allemaal n keer gebruiken, is het nuttig om eerst van de rijen van A de som uit (b) te berekenen:

$$\text{Rijterm}_i = \sum_{j=1}^{n/2} A_{i,2j-1} \cdot A_{i,2j}$$

en evenzo voor de kolommen van B. Als n oneven is, dan wordt voor elk rij-kolom produkt de laatste term apart meegenomen.
Schrijf een functie die m.b.v. deze methode twee matrices vermenigvuldigt.

(III) **Lineaire vergelijkingen.**
Schrijf een functie die de vergelijking

$$\mathbf{Ax} = \mathbf{b}$$

oplost, met $\mathbf{A}$ een $N \times N$ matrix en $\mathbf{b}$, $\mathbf{x}$ N-dimensionale vectoren. Doe dit m.b.v. de rij-veeg techniek. Deze maakt gebruik van het gegeven dat de oplossing onveranderd blijft als we links en rechts van het =-teken twee rijen verwisselen of één rij vermenigvuldigd met een bepaalde factor van een andere aftrekken. Eerst maak je met behulp hiervan de eerste elementen van rij 2 tm. N gelijk aan nul:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \rightarrow \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a'_{22} & \cdots & a'_{2n} \\ \vdots & & & \\ 0 & a'_{n2} & \cdots & a'_{nn} \end{pmatrix} = \mathbf{A}'$$

Waarbij a'_{ij} is ontstaan door de eerste rij met een bepaalde factor van de $i-$de rij af te trekken. Dezelfde lineaire combinaties voer je uit met $\mathbf{b}$. Daarna doe je hetzelfde met de tweede kolom etc. Als a'_{22} nu nul blijkt te zijn, dan kun je niet meer verder vegen. In dat geval moet je twee rijen van $\mathbf{A}$ en de ovreenkomstige elementen van $\mathbf{b}$ verwisselen zodat a'_{22} niet meer nul is. Uiteindelijk houd je dan een bovendriehoeks matrix $\mathbf{U}$ over:

$$\mathbf{U} \cdot \mathbf{x} = \mathbf{b}'$$

met $u_{ij} = 0$ als $i > j$. De oplossing $\mathbf{x}$ wordt nu gegeven door

$$x_i = \frac{1}{u_{ii}} \left(b'_i - \sum_{j=i+1}^{N} u_{ij} x_j \right)$$

(IV) **Determinanten.**
Gebruik de in het vorige onderdeel beschreven veegmethode in een functie die de determinant van een $N \times N$ matrix $\mathbf{A}$ uitrekent.

10. Sorteren.

In deze opgave gaan we volgens twee (vrij langzame) rechttoe-rechtaan methoden een rij getallen sorteren. In een volgende opgave coderen we het snelle "Quicksort" algoritme.
De getallen, die met een randomgenerator gemaakt zijn, worden ingelezen uit een file.

(a) **Selection sort.**
We gebruiken eerst de simpelste methode: loop door de rij op zoek naar het grootste element. Zet dit element dan achteraan. Vervolgens begin je opnieuw, maar nu hoef je de rij slechts te doorlopen tot het voorlaatste element; zo ga je door totdat alles gesorteerd is.
Geef een schatting voor het aantal stappen dat nodig is in dit algoritme.

(b) **Bubblesort.**
Nu gebruiken we een methode waarbij je door de rij loopt van het eerste naar het voorlaatste element en telkens als een element groter is dan het volgende dan verwissel je beide. Vervolgens begin je weer opnieuw.
Geef weer een schatting voor het aantal stappen dat dit algoritme gebruikt.

11. Simulatie.

Je hebt een quiz gewonnen bij RTL4! De prijs is een auto of een afschuwwekkend slaapkamerlampje. De quizmaster heeft drie doosjes. Als je het goede doosje aanwijst dan heb je de auto, anders het lampje. Je wijst een doosje aan, waarna de quizmaster een ander doosje opent, dat leeg is (dus niet de auto oplevert). Nu vraagt hij aan je of je nog op je eerste keuze terug wilt komen.

(a) Verhoog je de kans op de auto door het andere, nog niet geopende doosje te nemen?

(b) Controleer dit door een computersimulatie te schrijven waarin N maal dit spel gespeeld wordt (met bijv. $N = 1000$).
De doosjes krijgen de nummers 0, 1 en 2. Je kunt de random generator gebruiken om een doosje te kiezen waar de auto in zit. Dan kies je met de random generator één van de drie doosjes. Vervolgens gebruik je de random generator om de quizmaster één van de overige twee te openen mits die leeg is. Dan verander je de oorspronkelijke keuze. Conclusie?

12. **Postkantoor.**
Je hebt het gebracht tot directeur van een postkantoor. Soms denk je met heimwee terug aan de tijd dat je Programmeren deed en je komt op het idee om de gang van zaken in het postkantoor op je PC te simuleren. Er zijn P loketten open. In een bepaalde tijd T (bijv. één dag) komen er N klanten. Als er een klant binnenkomt, dan gaat hij bij het loket met de kortste rij staan. De handelingen aan de loketten vergen verschillende tijden. Het gemiddelde van deze tijden noemen we τ. De bedoeling is om uit te vissen hoeveel mensen er gemiddeld in een rij staan.

Omdat de komst van de klanten (naar je mag aannemen) ongecorreleerd is (d.w.z. dat de komst van een nieuwe klant niet afhangt van het feit of er kort tevoren een andere klant is geweest), zijn de mogelijke aankomsttijden volkomen random. Je begint dus met het vullen van een array (ter lengte N) met de aankomsttijden van de klanten. Vervolgens worden deze aankomsttijden gesorteerd, zodat de eerste klant op plaats 1 in de array staat. De eerste klant gaat naar loket 1.

Zodra hij hier gearriveerd is, bepaal je met behulp van een functie `GeefLoketTijd` de tijd die aan dit loket nodig is. Deze functie levert een random getal volgens de verdeling:

$$P(t) \propto te^{-t^2}.$$

Het gemiddelde van deze verdeling is $\sqrt{\pi}/2$. Je krijgt deze verdeling door een random getal x tussen 0 een 1 te transformeren volgens:

$$t = \sqrt{-\ln(x)}.$$

Dan heeft t de gewenste verdeling. Om nu een verdeling van deze vorm met een gemiddelde τ te krijgen, vermenigvuldig je t met $2\tau/\sqrt{\pi}$.

Er hebben telkens *gebeurtenissen* plaats: òf er komt een nieuwe klant binnen òf er is er een geholpen en deze vertrekt. Na elke gebeurtenis moet je nagaan wat de volgende gebeurtenis is, d.w.z. of er een klant binnenkomt of dat er een vertrekt en in dit laatste geval aan welk loket dit gebeurt. Als een klant vertrekt aan loket i, dan moet (als er nòg een klant aan dit loket staat) voor dit loket met bovenstaande functie weer een nieuw tijdstip worden gegenereerd waarop de klant vertrekt. Op elk moment is er dus minimaal één gebeurtenis mogelijk (komst van een klant, alle loketten zijn vrij) en maximaal $P+1$ (vertrek van één van de P loketten of komst van een klant). Na elke gebeurtenis zijn er dus tussen 1 en $P+1$ nieuwe gebeurtenissen mogelijk die allen op bekende tijden zullen plaatsvinden. Welke de eerstvolgende gebeurtenis is, vind je door het minimum van deze tijden te zoeken.

Bepaal zo hoeveel klanten er gemiddeld geholpen kunnen worden. Hiervoor moet je het programma een aantal malen laten lopen.

13. **Differentiaalvergelijkingen.**
Zeer veel fysische problemen worden geregeerd door een differentiaalvergelijking. In deze opdracht is het de bedoeling om zulke vergelijkingen op verschillende manieren numeriek op te lossen. Het resultaat dient op het scherm te worden getekend.

(a) We beginnen met de meest directe methode die we zullen toepassen op een eerste orde differentiaalvergelijking:

$$\begin{aligned} \tfrac{dx(t)}{dt} &= f(x) \\ x(t_0) &= x_0 \end{aligned}$$

Gevraagd wordt nu de waarde van $x(t)$ voor $t_0 \le t \le t_1$, voor zekere t_1.
De eenvoudigste schatting voor $x(t_0 + \Delta)$ krijgen we door aan te nemen dat $f(x)$ tussen t_0 en $t_0 + \Delta$ niet noemenswaardig verandert; dit geeft:

$$x(t_0 + \Delta) = x(t_0) + f(x_0) \cdot \Delta$$

Het zal duidelijk zijn dat deze benadering beter werkt naarmate Δ kleiner wordt.
Met de gevonden waarde voor $x(t_0+\Delta)$ bepalen we $x(t_0+2\Delta)$ enz., totdat we bij $x(t_1)$ zijn aangeland.
Schrijf een procedure die de differentiaalvergelijking op deze wijze oplost. Definieer f als een functie:

```
float f (float x)
```

De procedure heeft als parameters de linker- en rechtergrens van het interval waarover geïntegreerd wordt, de beginvoorwaarde x_0 en de stapgrootte Δ.

Gebruik deze procedure om de oplossing van de volgende beginwaardeproblemen op het scherm te tekenen:

$$\begin{array}{ll} \frac{dx(t)}{dt} = -x(t), & x(0) = 1 \quad t \in [0,1] \\ \frac{dx(t)}{dt} = \sqrt{(1-x^2(t))}, & x(0) = 0 \quad t \in [0,\pi/2]. \end{array}$$

Controleer het gevonden resultaat.

(b) Met behulp van een soortgelijke methode als in het vorige onderdeel kunnen we tweede orde differentiaalvergelijkingen aan.
Schrijf een procedure die het volgende beginwaardenprobleem oplost:

$$\begin{array}{l} \frac{d^2x(t)}{dt^2} = f(x) \\ x(t_0) = 0; \qquad \frac{dx(t_0)}{dt} = x_0'. \end{array}$$

Test deze procedure door bijvoorbeeld f constant te nemen.

(c) Los nu het probleem uit het vorige onderdeel op m.b.v. het Verlet-algoritme. Dit werkt als volgt.
Een benadering van $x(t+\Delta)$ is:

$$x(t+\Delta) = x(t) + \Delta\dot{x}(t) + \frac{\Delta^2}{2}\ddot{x}(t) + \ldots$$

Evenzo:

$$x(t-\Delta) = x(t) - \Delta\dot{x}(t) + \frac{\Delta^2}{2}\ddot{x}(t) + \ldots$$

dus

$$x(t+\Delta) + x(t-\Delta) = 2x(t) + \Delta^2 f(x(t)) + \ldots.$$

Dus

$$x(t+\Delta) = 2x(t) - x(t-\Delta) + \Delta^2 f(x(t)) + \ldots.$$

Je gebruikt dus $x(t)$ en $x(t-\Delta)$ om $x(t+\Delta)$ te vinden.

(d) Gebruik de Verlet procedure om de beweging van een deeltje in een krachtenveld:

$$\frac{d^2x}{dt^2} = F_x(x,y)$$

$$\frac{d^2y}{dt^2} = F_y(x,y)$$

te bepalen.
Neem $\mathbf{F} = -\frac{\mathbf{r}}{r}$, d.w.z. een deeltje in een trechter, en teken de baan van dit deeltje op het scherm.

(e) In het geval van een lineaire differentiaalvergelijking

$$\frac{d^2x(t)}{dt^2} = f(t)x(t)$$

kunnen we het Verlet algoritme nog verbeteren. De verbeterde methode is ontwikkeld door Numerov.
Merk op dat

$$x(t+\Delta) + x(t-\Delta) = 2x(t) + \Delta^2 x^{(2)}(t) + \frac{\Delta^4}{12}x^{(4)}(t) + \frac{\Delta^6}{360}x^{(6)}(t) + \ldots$$

Ga nu over op $w(t) = \left[1 - \frac{\Delta^2}{12}f(t)\right]x(t)$.
Dan:

$$w(t+\Delta) + w(t-\Delta) = 2w(t) + \Delta^2 x(t)f(t) - \frac{\Delta^6}{240}x^{(6)}(t) + \ldots$$

Dit geeft weer een verband tussen $x(t+\Delta)$, $x(t)$ en $x(t-\Delta)$, echter nu is de procedure goed tot op orde Δ^6 terwijl Verlet slechts goed is tot op orde Δ^4!
Schrijf een procedure om de lineaire differentiaalvergelijking op te lossen m.b.v. de Numerov methode.
Test deze procedure aan de hand van de Schrödinger vergelijking voor een deeltje in een rechthoekige potentiaal:

$$-\frac{\partial^2\psi}{dx^2} = (E - V(x))\psi(x)$$

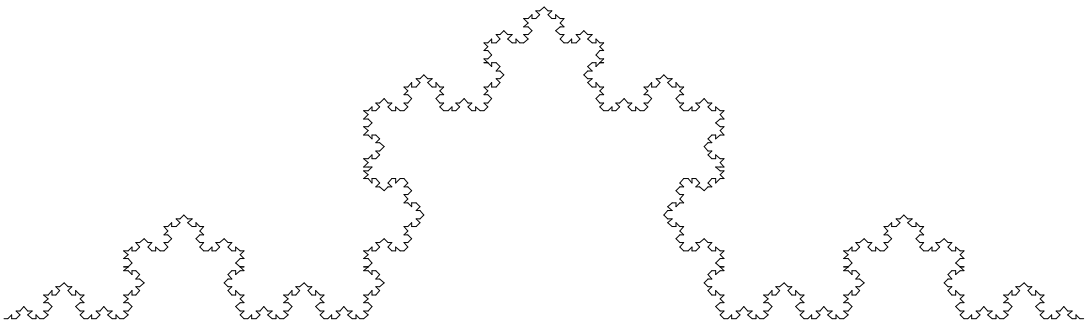
met $x \geq 0$, $\psi(x_0) = 0$ en

$$\begin{array}{ll} V(x) = -V_0 & 0 \leq x \leq a \\ V(x) = 0 & x \geq a \end{array}$$

(f) Oplossen Schrödinger vergelijking (Uit college Inleiding Quantum Mechanica)

14. Recursie

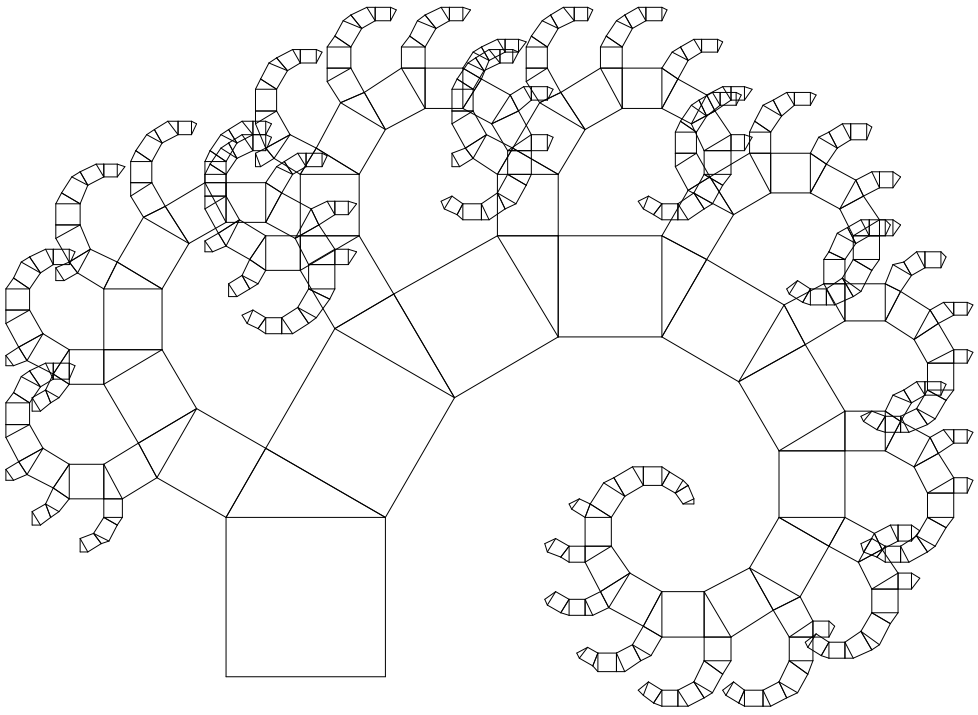
(a) Schrijf een recursieve procedure die het plaatje van figuur A.1 op het scherm tekent:



Figuur A.1: *Koch-kromme.*

Deze "Koch-kromme" is een voorbeeld van een zogenaamde fractal. Bij elke stap wordt precies één-derde uit het midden van een lijntje weggenomen en vervangen door twee lijntjes die samen met het weggehaalde stuk een gelijkzijdige driehoek vormen. Daarna wordt voor elk van de vier resulterende lijnstukjes weer hetzelfde procédé uitgevoerd. Als afbreekvoorwaarde neem je dat de lijntjes niet onder een minimale lengte uikomen.

(b) De zogenaamde "boom van pythagoras" is een figuur die is opgebouwd uit een "kiem", bestaande uit een vierkantje met daarop een driehoek met hoeken van 30, 60 en 90 graden. Door deze kiem op een bepaalde manier te herhalen, wordt het plaatje van figuur A.2 opgebouwd.



Figuur A.2: *Boom van Pythagoras.*

Schrijf een recursief programma dat dit plaatje op het scherm tekent.
Als afbreek-voorwaarde neem je dat de lengte van de zijde van het vierkant niet onder een minimale lengte komt.

(c) Schrijf een programma dat een rij sorteert volgens het "quicksort" algoritme.

(d) Een voorbeeld van een **percolatie**probleem is het volgende.
Beschouw een vierkant rooster. Stel je voor dat je over het rooster kunt lopen van een roosterpunt naar diens (vier) naaste buren. Er is echter een fractie p van de roosterpunten geblokkeerd, d.w.z. daar kun je niet naartoe stappen. Deze punten zijn random gekozen. In het programma kun je zo'n rooster maken m.b.v. de `rand` procedure:

`rand()` geeft een random integer getal tussen 0 en 32767.

Schrijf een programma dat *alle* paden zoekt die van de linkerzijde naar de rechterzijde van het rooster lopen.
Dit probleem is van groot economisch belang: van poreuze gesteenten waar olie inzit wil men graag weten of er lange kanalen zijn die met elkaar in verbinding staan.
Theoretische berekeningen hebben aangetoond dat voor een (oneindig groot) rooster er lange (d.w.z. ook oneindig grote) paden bestaan voor $p < 0.5$. Ga m.b.v. het programma na dat dit klopt.