Numerieke oplossing van zelf-organisatie in chemische systemen

Michiel Meeuwissen

31 januari 1997

Inhoudsopgave

Dit is een verslag behorende bij het vak *Modelvorming en Simulatie*, en de inhoud is zeer geïnspireerd door het boek van Koonin en een opgave die daarin staat [1].

1 Introductie

We beschouwen (en volgen [1, 2]) de volgende reacties tussen de stoffen A, B, D, E, X en Y. Waarin X en Y tussenprodukten zijn.

$$A \longrightarrow X \tag{1}$$
$$B + X \longrightarrow Y + D \tag{2}$$
$$2X + Y \longrightarrow 3X \tag{3}$$
$$X \longrightarrow E \tag{4}$$

We gaan er vanuit dat deze reacties allemaal maar 1 kant op lopen en dat de stoffen D en E verder niet meer reageren of worden afgevoerd uit het vat.

De concentraties van A en B worden constant in de tijd en ruimte gehouden en de ruimte is 1 dimensionaal in dit hele verslag.

We kunnen dan de volgende vergelijkingen opstellen voor de concentraties van de stoffen X en Y, welke ook X en Y heten:

$$\partial_t X = k_a A - k_b BX + k_c X^2 Y - k_d X + D_X \partial_x^2 X \tag{5}$$
$$\partial_t Y = k_b BX - k_c X^2 Y + D_Y \partial_x^2 Y \tag{6}$$

Deze vergelijkingen kunnen we in dimensieloze vorm schrijven (zie nog steeds [2]):

$$\partial_t X = A - (B+1)X + X^2 Y + D_X \partial_x^2 X \tag{7}$$
$$\partial_t Y = BX - X^2 Y + D_Y \partial_x^2 Y \tag{8}$$

We kunnen hieruit vinden wat X en Y in evenwicht zijn (X_0 en Y_0) door $\partial_t Q = 0$ en $\partial_x^2 Q = 0$ te kiezen. Q staat en zal staan voor (X of Y):

$$X_0 = A \qquad Y_0 = B/A \tag{9}$$

2 Lineaire stabiliteits analyse

We kunnen proberen te bekijken hoe de concentraties X en Y zich als functie van plaats en tijd gaat gedragen door kleine verstoringen van evenwicht te beschouwen:

$$X(x,t) = A + \delta_X(x,t) \tag{10}$$
$$Y(x,t) = B/A + \delta_Y(x,t) \tag{11}$$

En we vullen deze in in vergelijking 7 en 8 en we vinden eenvoudig:

$$\begin{aligned} \partial_t \delta_X &= (B-1)\delta_X + A^2\delta_Y + D_X \partial_x^2 \delta_X && (12) \\ \partial_t \delta_Y &= -B\delta_X - A^2\delta_Y + D_Y \partial_x^2 \delta_Y && (13) \end{aligned}$$

Waarin we kwadratische termen in δ_Q hebben verwaarloosd. De functies δ_X en δ_Y gaan we zo meteen Fourier ontwikkelen, maar eerst gaan we een paar randcondities bedenken. Een voor de hand liggende is de 'No-Flux' randconditie. De stoffen X en Y kunnen aan de wanden niet stromen (er zitten daar immers al wanden), wat wil zeggen dat $\partial_x Q(x=0)=0$ en $\partial_x Q(x=1)=0$ (stroom is evenredig met ∂_x(concentratie)). We kunnen ook eisen dat de concentraties op de randen hun evenwichtswaarde aannemen.

We ontwikkelen dan δ_X en δ_Y:

$$\delta_Q(x,t) = \sum_{m=0}^{\infty} \delta_{Qm} e^{\omega_m t} \cos(m\pi x) \tag{14}$$

Zo is aan de No-Flux voorwaarde automatisch voldaan. Als we de cosinus door een sinus vervangen dan hebben we de andere randvoorwaarde. Maar dit maakt voor het vervolg niet uit.

Invullen in de vergelijkingen 12 en 13 geeft:

$$\begin{aligned} \sum_m \delta_{Xm}\omega_m e^{\omega_m t}\cos(m\pi x) &= (B-1)\sum_m \delta_{Xm} e^{\omega_m t}\cos(m\pi x) \\ &\quad + A^2 \sum_m \delta_{Ym} e^{\omega_m t}\cos(m\pi x) \\ &\quad - D_X \sum_m \delta_{Xm} e^{\omega_m t}\cos(m\pi x)(m\pi)^2 && (15) \end{aligned}$$

En een analoge vergelijking voor δ_{Ym}. Deze vergelijkingen moeten gelden voor alle keuzes van $\{\delta_{Qm}\}$ en dus voor elke m afzonderlijk (de cosinus en de e-macht term vallen dan dus weg):

$$\begin{aligned} \omega_m \delta_{Xm} &= (B-1)\delta_{Xm} + A^2\delta_{Ym} - D_X(m\pi)^2\delta_{Xm} && (16) \\ \omega_m \delta_{Ym} &= -B\delta_{Xm} - A^2\delta_{Ym} - D_Y(m\pi)^2\delta_{Ym} && (17) \end{aligned}$$

Alle termen in deze twee vergelijkingen zijn van orde δ_{Qm} en de vergelijkingen zijn dus afhankelijk. Ze hebben slechts een gezamenlijke oplossing voor $\{\delta_{Xm}, \delta_{Ym}\}$ als:

$$\omega_m^2 + (1-B+d_X+A^2+d_Y)\omega_m + (A^2+d_Y)(1-B+d_X) + BA^2 = 0 \tag{18}$$

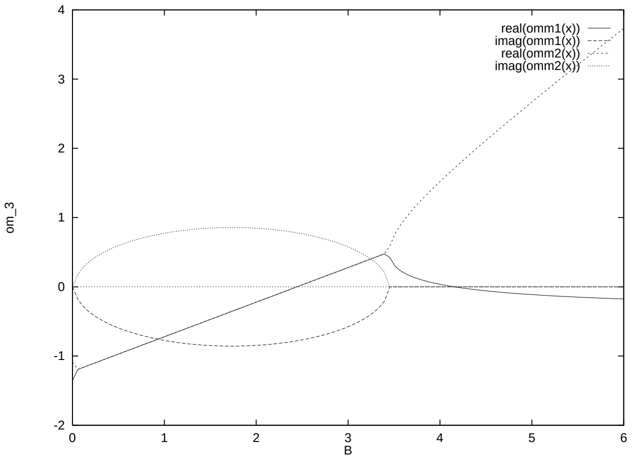
met $d_Q = D_Q(m\pi)^2$. We definiëren verder nog $\alpha_m = 1-B+d_X$ en $\beta_m = A^2+d_Y$ en we krijgen:

$$\omega_{m\pm} = -\frac{1}{2}\left(\alpha_m + \beta_m \pm \sqrt{(\alpha_m-\beta_m)^2 - 4BA^2}\right) \tag{19}$$

Uit vergelijking 14 zien we dat negatieve waarden voor ω_m corresponderen met in de tijd gedempte oplossingen. Positieve waarden voor ω_m zouden juist kleine verstoringen versterken. Als ω_m imaginair wordt dan gaat de oplossing oscilleren, en dat wordt daarbij eventueel nog vesterkt of gedempt al naar gelang het reëele deel.

We kunnen nu dus kijken bij welke m en welke ω_m welk gedrag optreedt.

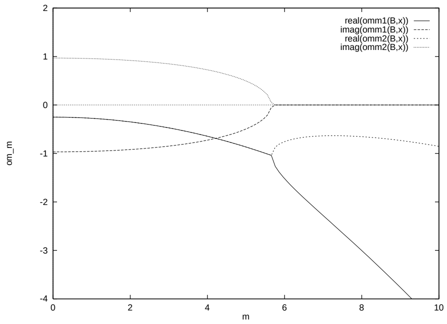
Jammer genoeg zijn er tamelijk veel variabelen. We kiezen er dus maar eerst een paar vast, zoals ook in [1]. We kiezen $A = 1$, $D_X = 0,001$ en $D_Y = 0,004$ en we kijken hoe ω_3 van B afhangt. Het resultaat staat in figuur 1. We zien dat



Figuur 1: ω_3 als functie van B.

we al meteen een paar interesante oplossingen hebben. Bij $B < 2,5$ hebben we een negatief reëel gedeelte van de beide ω's en ook een imaginair gedeelte. We krijgen bij deze B's dus gedempte oscillerende oplossingen. Bij wat grotere B's hebben we nog steeds een imaginair stuk in de ω's maar nu een positief reëel deel. We krijgen nu dus groeiende oscillerende oplossingen. Bij nog grotere B (groter dan zo'n 3,5) wordt het imaginaire stuk van de ω 0 en krijgen we 2 reëele oplossingen. Eén daarvan is altijd positief en die oplossing zal dus op den duur gaan overheersen. In het hoofdstuk resultaten beschrijven we enige van deze oplossingen. In [2] zijn er voor de verschillende situaties wat algemenere formules afgeleid.

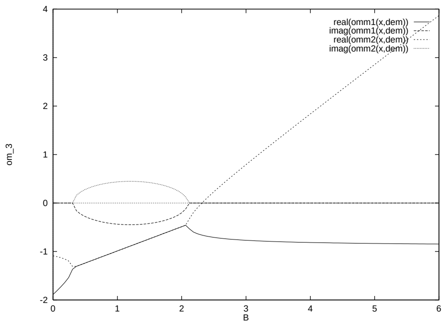
In het algemeen zullen de beginconcentraties van de stoffen niet slechts uit één m mode bestaan maar kan het vanalles zijn. Het zal dan dus een superpositie van alle m's zijn zoals in vergelijking 14. De afhankelijkheid van m voor een bepaalde instelling van de rest van de variabelen is getekend in figuur 2. We zien bij grotere waarden van m bijvoorbeeld ook de ontbrekende combinatie van een ω met een negatief reëel deel en een ontbrekend imaginair deel. Daar krijgen we dus een dempende, maar niet in de tijd oscillerende oplossing. In het algemeen zullen dus alle oplossingen, bij deze keuze voor A, B en D_Q, stabiel zijn, want



Figuur 2: $A = 1; D_X = 0,001; D_Y = 0,004; B = 1,5$

nergens heeft ω een positief reëel deel.

Dit kunnen we ook bereiken door bijvoorbeeld $D_Y = 0,01$ te kiezen, en verder dezelfde waarden voor A, D_X en $m = 3$. ω_3 als functie van B is dan uitgezet in figuur 3. Bij bijvoorbeeld tamelijk kleine B (zoals $B = 0,2$) zijn er



Figuur 3: $A = 1, D_X = 0,001, D_Y = 0,01$

nu dus ook stabiele niet oscillerende oplossingen.

De beschrijving van dit systeem gaat in [2] veel verder, en daarnaar wordt dus verder ook verwezen.

3 Beschrijving van het numeriek oplossen

Diffusie problemen kunnen we numeriek het beste oplossen met het 'Crank-Nicholson' algoritme (zie [3] hoofstuk 19.2). Dit heeft een zogenaamd FTCS (Forward Time Centered Space) schema. Het is het gemiddelde van een expliciete en een impliciete variant.

Voor de diffusie term geldt dan:

$$\left.\frac{Q_j^{n+1}-Q_j^n}{\Delta_t}\right|_{\text{diffusie}} = \frac{D_Q}{2}\left(\frac{(Q_{j+1}^{n+1}-2Q_j^{n+1}+Q_{j-1}^{n+1})+(Q_{j+1}^n-2Q_j^n+Q_{j-1}^n)}{\Delta_x^2}\right) \tag{20}$$

Waarin de index j dus voor positie en n voor tijd staat.

In dit geval komt er echter nog een extra term bij voor de veranderingen in de concentraties door de chemische reacties. Uit de vergelijkingen 7 en 8 zien we hoe deze termen er uit moeten zien:

$$\partial_t X|_{\text{chemisch}}(X,Y) = A-(B+1)X+X^2Y \tag{21}$$

$$\partial_t Y|_{\text{chemisch}}(X,Y) = BX-X^2Y \tag{22}$$

Hierin zitten geen ruimtelijke afgeleiden, slechts tijds-, en we kunnen in het algemeen dus als volgt discretiseren:

$$\left.\frac{Q_j^{n+1}-Q_j^n}{\Delta_t}\right|_{\text{chemisch}} = a\,\partial_t Q|_{\text{chemisch}}(X_j^n,Y_j^n)+b\,\partial_t Q|_{\text{chemisch}}(X_j^{n+1},Y_j^{n+1}) \tag{23}$$

waarin a voor de expliciete en b voor de impliciete term staat.

We kiezen alleen de expliciete term, anders wordt de vergelijking kwadratisch impliciet, en veel moeilijker:

$$a=1, b=0 \tag{24}$$

Zo krijgen we namelijk slechts een verzameling lineaire impliciete vergelijkingen voor Q^{n+1}:

$$\begin{aligned}\frac{Q_j^{n+1}-Q_j^n}{\Delta_t} &= \left.\frac{Q_j^{n+1}-Q_j^n}{\Delta_t}\right|_{\text{diffusie}}+\left.\frac{Q_j^{n+1}-Q_j^n}{\Delta_t}\right|_{\text{chemisch}}\\ &= \frac{D_Q}{2}\left(\frac{(Q_{j+1}^{n+1}-2Q_j^{n+1}+Q_{j-1}^{n+1})+(Q_{j+1}^n-2Q_j^n+Q_{j-1}^n)}{\Delta_x^2}\right)\\ &\quad+\partial_t Q|_{\text{chemisch}}(X_j^n,Y_j^n)\end{aligned} \tag{25}$$

Als we b ongelijk 0 hadden gekozen worden de vergelijkingen niet-lineair inQ^{n+1} want de chemische termen zijn niet lineair in Q. Dan wordt alles veel ingewikkelder en dit geval is niet bekeken.

We kunnen voor het oplossen van deze vergelijkingen het 'tridag' algoritme uit Numerical Recipes gebruiken. We vinden namelijk

$$\begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot & \cdot \\ & & & & a_N & b_N \end{pmatrix} \cdot \begin{pmatrix} Q_1^{j+1} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ Q_N^{j+1} \end{pmatrix} = \begin{pmatrix} r_1^j \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ r_N^j \end{pmatrix} \tag{26}$$

a, b, c en r in deze vergelijking volgen uit vergelijking 25. Het blijkt dat 'vectoren' a en c hetzelfde kunnen zijn en in het programma bestaat dus c niet, maar wordt daarvoor a gebruikt. De randvoorwaarden kan men in b_1, a_1, a_N en b_N omschrijven wat in het programma dan ook gebeurt (zie appendix B).

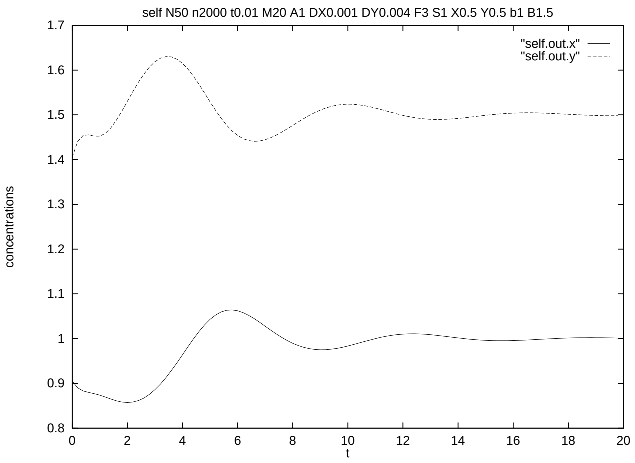
r is een functie van X^j zoals ook in het programma is omschreven en zoals eenvoudig volgt uit de bovenstaande vergelijkingen.

4 Resultaten

We kunnen het programma gebruiken om enkele van de speciale eigenschappen van de beschreven reacties te bekijken. Enkele resulaten hiervan worden in dit hoofstuk beschreven.

4.1 Een simpel oscillerende dempende oplossing.

Zoals eerder beschreven (figuur 1) zouden we bij $A = 1; D_X = 0,001; D_Y = 0,004; m = 3$ en $B = 1,5$ een oscillerende dempende oplossing moeten krijgen. Het programma is geprobeerd met deze waarden en we krijgen in het midden van het vat een concentratie verloop van de 2 stoffen zoals in figuur 4. Bij deze

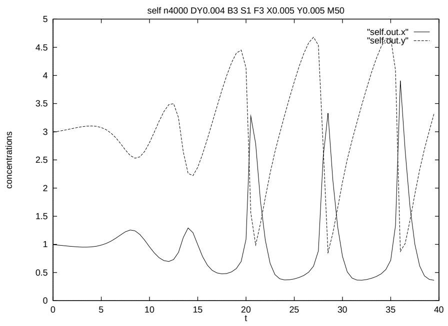


Figuur 4: Concentraties van de stoffen X en Y als functie van de tijd, met $A = 1; D_X = 0,001; D_Y = 0,004; B = 1,5$ en $m = 3$

instellingen vinden we $\mathrm{Im}(\omega) = \pm 0,85$ en dus een periode van $2\pi/\mathrm{Im}(\omega) = 7,41$. Dit klopt aardig met wat we zien in figuur 4. Als we een filmje maken zien we inderdaad de twee concentraties wat oscilleren en dan beiden uiteindelijk naar de evenwichtswaarde lopen.

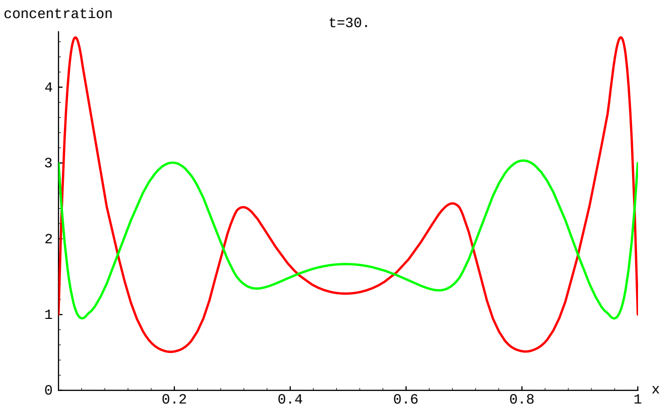
4.2 Een groeiend oscillerende oplossing

Bij dezelfde waarden als net, alleen voor B kiezen we nu 3, krijgen we, zoals we al gezien hebben, een oscillerend groeiende oplossing. We kijken weer op een bepaald ruimtelijk punt (in het midden) om de tijdsafhankelijkheid te bestuderen. We zien het resultaat in figuur 5. Theoretisch zou men een frequentie moeten vinden van 10,8. We zien dat, zolang de amplitudes niet zeer groot worden, dit wel ongeveer klopt. Er is nu begonnen met een 2 sinussen voor de concentraties van X en Y met een zeer kleine amplitude (0,005). We kijken nu



Figuur 5: Concentraties van de stoffen X en Y als functie van de tijd, met $A = 1; D_X = 0,001; D_Y = 0,004; B = 3; m = 3$ en $x = 0,5$

bij een bepaalde t ook naar de ruimtelijke verdeling van de stoffen want deze is niet homogeen en ook niet meer sinus-vormig. In het Mathematica-filmje was te

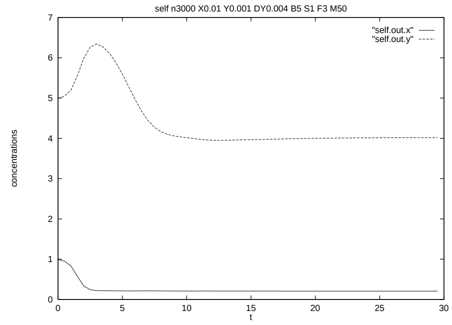


Figuur 6: Concentraties van de stoffen X en Y als functie van de tijd, met $A = 1; D_X = 0,001; D_Y = 0,004; B = 3; m = 3$ en $t = 30$

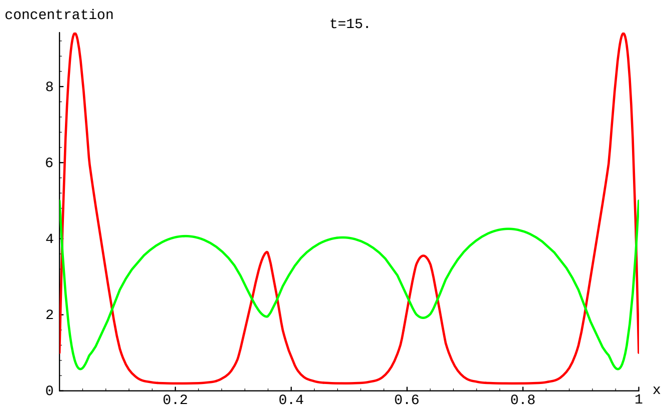
zien dat vooral in het midden we wilde concentratie fluctuaties krijgen, terwijl de pieken aan de wanden ongeveer in de tijd constant blijven. Bij de andere randvoorwaarde (afgeleiden op de rand nul) vinden we dat het overal sterk gaat oscilleren. De ruimtelijke verdeling is zeker niet triviaal en kam erg ingewikkeld worden. In dit geval bleek er echter wel een bepaalde periodiciteit in te zitten, en komen dezelfde structuren steeds terug.

4.3 Niet oscillerende groeiende oplossingen

Bij $B = 5$ vinden we dan dat het imaginaire deel van ω_m 0 is en we hebben een positief reëel deel. De oplossingen zullen dus groeien in de tijd, maar niet gaan oscilleren.



Figuur 7: Concentraties van de stoffen X en Y als functie van de tijd, met $A = 1; D_X = 0,001; D_Y = 0,004; B = 5; m = 3$ en $x = 0,5$

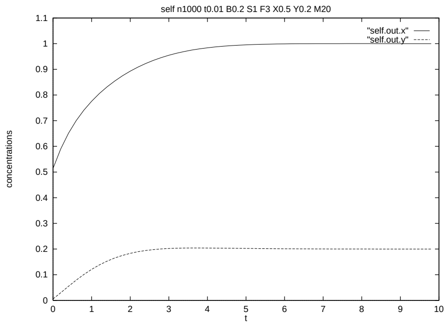


Figuur 8: Concentraties van de stoffen X en Y als functie van de plaats, met $A = 1; D_X = 0,001; D_Y = 0,004; B = 5; m = 3$ en $t = 15$

Inderdaad vinden we uit een zeer kleine sinusvormige verstoring van het evenwicht grote schommelingen in de plaats, maar het gaat er zonder oscilleren naar toe (zoals men mooi kon zien in het mathematica filmpje) en blijft daarna constant, in tegenstelling tot het vorige geval. Verwezen wordt naar figuur 7 en figuur 8.

4.4 Dempend niet-oscillerende oplossing

De laatste interessante oplossing die we analytisch konden voorzien is een dempende niet-oscillerende oplossing, met een ω zonder imaginair deel en een negatief reëel deel. Dit is gedaan, en het resultaat staat in figuur 9. In een filmje zijn



Figuur 9: Concentraties van de stoffen X en Y, met $A = 1; D_X = 0,001; D_Y = 0,01; B = 0,2$

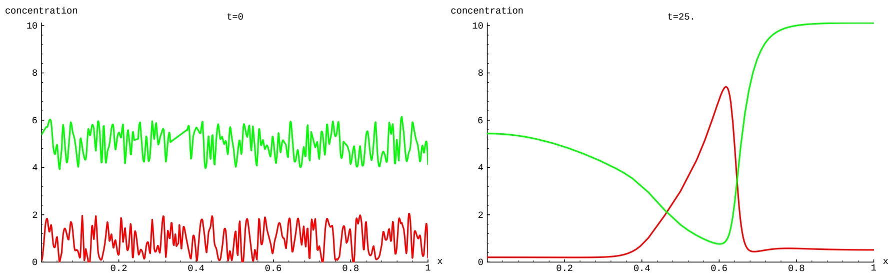
de twee sinusvormige concentraties te zien die naar een homogeen evenwicht veranderen. Bij de andere randvoorwaarde (voor de afgeleiden) vinden we vergelijkbaar gedrag. Ook bij bijvoorbeeld $D_Y = 0,004$ en $B = 1,5$ is er een dergelijk resultaat gevonden. Zoals ook was voorspeld met figuur 3.

4.5 Samengestelde oplossingen

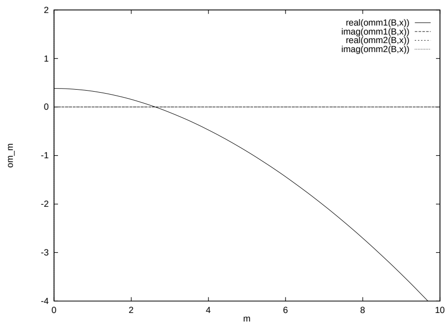
Als de verstoringen van het evenwicht niet meer sinusvormig zijn, krijgen we volgens de theorie superposities van oplossingen.

4.5.1 Random verstoring

Een random verstoring van de concentraties rond het evenwicht kan zeer bizarre oplossingen geven, maar ook erg vaak is het resultaat dat de beide concentraties gewoonweg wat blijven schommelen rond het evenwicht in plaats en tijd. We geven hier een voorbeeld van een ander geval in figuur 10. De structuur die in het rechter plaatje te zien is beweegt naar rechts en als hij helemaal weg is komt er links weer een nieuwe. Dit gaat tenminste 3 keer zo door (verder is niet gekeken). Er kon voorspeld worden dat er misschien wel iets interesants zou kunnen gebeuren want het equivalent van figuur 2 heeft in dit geval wel bij bepaalde m een positief reëel deel (zie figuur 11). Er is geprobeerd of dit effect nagedaan kan worden door andere initiële waarden van de concentraties te kiezen. Dat blijkt inderdaad te kunnen en we kunnen zo ook een structuur de



Figuur 10: $A = 1; D_X = 0,001; D_Y = 0,004; B = 5$ en de concentraties fluctueerde op $t = 0$ met een amplitude van 1 rond de evenwichten. Er zijn 200 ruimtelijke punten.



Figuur 11: $A = 1; D_X = 0,001; D_Y = 0,004; B = 5$

andere kant op laten lopen (gebruik de opties S3(naar rechts lopen) en S4(naar links lopen) van programma, en zie bij de function initXfun en initYfun voor de omschrijving).

Bij $B = 1,5$ vonden we dat zulke random fluctuerende concentraties allebei naar evenwicht gingen, zoals ook was voorspeld met figuur 2.

4.6 Conclusies en Verklaringen

Het beschreven programma werkt goed en de theoretische voorspelling kwamen goed uit. Er is alleen nog geen verklaring gegeven voor het feit dat oplossingen die theoretisch zouden 'exploderen' dat in werkelijkheid niet doen. Dit is echter logisch, want concentraties kunnen niet negatief worden. De concentraties X en Y zijn namelijk relateerd en als de 1 toeneemt neemt de ander af. Het is logisch dat de toename van de één en afname van de ander niet als maar door kan gaan. De theorie voorspelt dit soort zaken niet omdat deze slechts gaat over kleine verstoringen, de kwadratische termen waren immers verwaarloosd.

De explosieve, maar niet oscillerende, oplossingen krijgen ook een ruimte structuur in de concentraties maar deze is constant in de tijd. Het is moeilijk om voor de ruimtelijke structuur een goede begripgevende verklaring te geven. Het is zo dat op elk punt er een evenwicht onstaat tussen wat er chemisch onstaat en wat er door diffusie wegstroomt.

We zien echter wel dat we met zo'n numeriek programma ook alles kunnen berekenen en we krijgen ermee een inzicht wat er verder gebeurt als de theoretische afleiding, immers alleen voor kleine fluctuaties, niet meer geldig is. Het daarom dus een waardevolle bijdrage daarop.

A Het type 'field'

In het header file 'fields.h' wordt het type 'field' gedifinieerd. Dit is een soort vector wat echter variabele grootte heeft en ook een offset. Deze offset is geinspireerd door Numerical Recipes, maar is eigenlijk onzin. Het heet field (een veld), omdat het meestal een discretisatie is van een 1 dimensionaal veld. Met veld wordt dan bedoelt datgene wat er ook mee bedoelt wordt in 'electrisch veld'.

De implementatie volgt:

```
//

/* Fields.h

   Michiel Meeuwissen

   december 1996

   In this the type 'field' is defined.
   A field is 1-offset 'floatt' array with variabel length.

*/

#include<stdio.h>
#include<stdlib.h>
#include<stdarg.h>

class field
{ private:
   floatt *values;
   floatt *ovalues; // with offset
```

```
  public:
   unsigned size;

   field(unsigned n=0);
   field(unsigned, floatt);
   field(unsigned, floatt f(floatt));
   // field(field &); // copy constructor
   ~field() { delete[] values; /* printf("destr. \n");*/}

   void resize(unsigned);
   void resize(unsigned, floatt);

   floatt &operator[](const long unsigned n)
      { // if(n>size || n < 1) { printf(" wrong n-value ..\n");return values[n%size];}
        return ovalues[n];
      }
   field operator+=(field);
   field operator+(field);
   field operator=(field);

   void write() const;
   void set(unsigned, ...);

   friend void writefields(unsigned, ...);
   friend unsigned maxsize(unsigned, ...);
   friend void toggle(field&, field&);
};

void tridag (field &a, field &b, field &c, field &r, field &u)
// algorithm taken from NR.
{ unsigned long n = a.size; // geen controles.
  unsigned long j;
  floatt bet;
  field gam(n);
  // writefields(4,a,b,c,r);
  if (b[1]==0) { printf("(tridag) b[1]==0, rewrite your equations.\n"); return;}

  u[1]=r[1]/(bet=b[1]);
  for(j=2; j<=n; j++) // Decomposition and forward substitution.
  { gam[j]=c[j-1]/bet;
    bet=b[j]-a[j]*gam[j];
    if (bet==0) { printf("(tridag) algorithm fails\n"); return;}
    u[j]=(r[j] - a[j]*u[j-1]) / bet;
  }
  for(j=(n-1); j>=1; j--)  // Backsubstitution
     u[j] -= gam[j+1]*u[j+1];
}// tridag
// getest en gecontroleerd, maar de bet==0 errormelding kan makkelijk verschijnen


field::field(unsigned n)
{ values=new floatt[n];
  size =n;
  // printf("constr. 1\n");
  if(values==NULL) {printf("(field) not enough memory...\n");
                    exit(1);
                   }
  ovalues=values-1;
}

void field::resize(unsigned n)
{ floatt *help;
  help = new floatt[n];
  if(help==NULL) {printf("(resize) not enough memory...\n");
                  exit(1);
                 }
  for(int i=0; i<(size<n ? size : n); i++) help[i]=values[i];
  delete values;
  size=n;
  values=help;
  ovalues=values-1;
}
void field::resize(unsigned n, floatt f)
{ delete values;
  values = new floatt[n];
  if(values==NULL) {printf("(resize) not enough memory...\n");
                    exit(1);
                   }
  for(int i=0; i<n; i++) values[i]=f;
  size=n;
  ovalues=values-1;
}


field::field(unsigned n, floatt f)
{ values=new floatt[n];
  size =n;
  // printf("constr. 2\n");
  if(values==NULL) {printf("(field) not enough memory...\n");
                    exit(1);
                   }
```

```
  ovalues=values-1;
  for (unsigned i=1; i<=size; i++) ovalues[i]=f;

}

field::field(unsigned n, floatt f(floatt))
{
  values=new floatt[n];
  size =n;
  // printf("constr. 3\n");
  if(values==NULL) {printf("(field) not enough memory...\n");
                    exit(1);
                   }
  ovalues=values-1;
  for (unsigned i=1; i<=size; i++) ovalues[i]=f((floatt)i/size);
}

/*
field::field(field &f)
{ size = f.size;
  values = new floatt[size]
  if(values==NULL) {printf("(field) not enough memory...\n");
                    exit(1);
                   }
  ovalues=values-1;
  for (unsigned i=1; i<=size; i++) ovalues[i]=f.ovalues[i];
}
*/

void field::write() const
{ for (unsigned i=1; i<= size; i++) printf("%u: %e \n",i, ovalues[i]);
}


void field::set(unsigned m, ...)
{ va_list argp;
  va_start(argp,m);
  for(unsigned i=1; i<=m; i++) ovalues[i]= va_arg(argp,double);
// met 'float' werkt dit absoluut niet.
  va_end(argp);
}


void testva_arg(unsigned n, ...)
{ va_list argp;
  va_start(argp,n);
  while (n--) printf("test: %e \n", va_arg(argp,double));
  va_end(argp);
}

unsigned maxsize(unsigned n, ...)
{ va_list argp;
  unsigned m=0;
  unsigned hulp;
  va_start(argp,n);
  while (n--) { hulp = va_arg(argp,field).size;
                if(hulp > m) m=hulp;
              }
  va_end(argp);
  return m;
}

void toggle(field &a, field &b)
{ floatt *hulp1, *hulp2;
  hulp1=a.values;
  hulp2=a.ovalues;
  a.values=b.values;
  a.ovalues=b.ovalues;
  b.values=hulp1;
  b.ovalues=hulp2;
}



void writefields(unsigned n, ...)
{ va_list argp;
  unsigned ms=0;
  unsigned hulp,m=n;
  field *f;
  va_start(argp,n);
  while (m--) { hulp = va_arg(argp,field).size;
                if(hulp > ms) ms=hulp;
              }
  for (unsigned i=0; i<ms; i++)
    { m=n;
      va_start(argp,n);
      printf("%u: ",i+1);
      while (m--) { f=&va_arg(argp,field); // ondiepe kopie.
                    f->size > i ? printf("%e ",f->values[i]) : printf(" xxxxxxxxxxxx ");
                  }
      va_end(argp);
```

```
      printf("\n");
    }

}
//
```

B Het programma

Het programma dat het tridag algoritme uitvoert en zorgt voor de begincondities volgt:

```
//

/* Self-organisation in chemical reactions

   Michiel Meeuwissen

   january 1997
*/

#define floatt float
#define maxfloatt 1e10
#include "fields.h"
#include <string.h>

enum boolean {False, True};

inline floatt dXdt(floatt x, floatt y, floatt a, floatt b)
 { return a + (x * y - 1 - b) * x;}
inline floatt dYdt(floatt x, floatt y, floatt b)
 { return (b - x * y) * x;}


// we define the basic variables global:
unsigned N=40; // size of grid
field X, Y, Xnew, Ynew;

floatt A=1., B=1.;
floatt initX=0.1, initY=0.1;
floatt l=1.;         // size of container
floatt dx=l/N;
floatt DX=0.001;        // diffusion coefficients.
floatt DY=0.001;
floatt   dt=0.01;      // time step
unsigned nit=1;       // number of iteration.
unsigned boundary=2;
unsigned startfun=0;
floatt freq=1;
// some dependent:

// these are the a and b which will be used in tridag (c=a)
field aX, bX, aY, bY;
floatt rX1, rX2, rY1, rY2;

field rX,rY;

void resizeglobals()
{ X.resize(N); Y.resize(N);
  Xnew.resize(N); Ynew.resize(N);

  aX.resize(N); bX.resize(N);
  aY.resize(N); bY.resize(N);

  rX.resize(N); rY.resize(N);
}

// set boundary conditions.
void setBC(int b)
{
// BC: X[1]-X[2]=0;
switch(b)
{
case 1: bX[1]=1; aX[1]=-1; rX[1]=0;
        bX[N]=1; aX[N]=-1; rX[N]=0;

        bY[1]=1; aY[1]=-1; rY[1]=0;
        bY[N]=1; aY[N]=-1; rY[N]=0;
        break;
case 2: bX[1]=1; aX[1]=0; rX[1]=A;
        bX[N]=1; aX[N]=0; rX[N]=A;

        bY[1]=1; aY[1]=0; rY[1]=B/A;
        bY[N]=1; aY[N]=0; rY[N]=B/A;
        break;
case 3: bX[1]=1; aX[1]=0; rX[1]=1;
        bX[N]=1; aX[N]=0; rX[N]=1;
```

```
        bY[1]=1; aY[1]=0; rY[1]=1;
        bY[N]=1; aY[N]=0; rY[N]=1;
        break;
default: printf("unknown boundary condition number specified.\n");
         exit(1);
}
}

floatt initXfun(floatt x, unsigned i=0)
  { switch(i)
   { case 0: return initX;
     case 1: return A + initX * ( boundary==2 ? sin(freq*3.14*x)
                                             : cos(freq*3.14*x));
     case 2: return A + 2*initX * ((floatt)rand()/RAND_MAX -0.5);
     case 3: return (x>0.33 && x<0.66) ? 5*(x-0.33)*A/0.33 : A/3;
     case 4: return (x>0.33 && x<0.66) ? 5*(1-(x-0.33)/0.33)*A : A/3;
     default: return initX;
   }
  }
floatt initYfun(floatt x, unsigned i=0)
  { switch(i)
    { case 0: return initY;
      case 1: return B/A + initY * ( boundary==2 ? sin(freq*3.14*x)
                                                 : cos(freq*3.14*x));
      case 2: return B/A + 2*initY * ((floatt)rand()/RAND_MAX-0.5);
      case 3: return (x<0.66) ? B*(1-x/0.66)/A : 2*B/A;
      case 4: return (x>0.33) ? B*(x-0.33)/(A*0.66) : 2*B/A;
      default: return initY;
    }
  }

void setglobals()
{ floatt aXval = -DX/(2*dx*dx), bXval = +DX/(dx*dx)+1/dt,
         aYval = -DY/(2*dx*dx), bYval = +DY/(dx*dx)+1/dt;
  for(int i=2; i<N; i++)
     { aX[i] = aXval; bX[i] = bXval;
       aY[i] = aYval; bY[i] = bYval;
     }
  for(int i=1; i<=N; i++)
    { X[i] = initXfun((floatt)i/N,startfun); Y[i] = initYfun((floatt)i/N,startfun);
    }

  rX1=1/dt - DX/(dx*dx), rX2=DX/(2*dx*dx);
  rY1=1/dt - DY/(dx*dx), rY2=DY/(2*dx*dx);

  setBC(boundary);
}


void itexpl()
{// valueing of r:
long unsigned i;

for(i=2; i<N; i++)
   { rX[i]=rX1*X[i] + rX2*(X[i+1] + X[i-1]) + dXdt(X[i],Y[i],A,B);
     rY[i]=rY1*Y[i] + rY2*(Y[i+1] + Y[i-1]) + dYdt(X[i],Y[i],B);
   }
tridag(aX,bX,aX,rX,Xnew);
tridag(aY,bY,aY,rY,Ynew);
//writefields(7,aX,bX,rX,X,Xnew,Y,Ynew);
toggle(X,Xnew);
toggle(Y,Ynew);

nit++;

}

boolean testXY()
{ boolean nok=False;
  for(int i=1; i<=N; i++)
     if((X[i]<0) || (Y[i]<0) || (X[i]>maxfloatt) || (Y[i]>maxfloatt)) nok=True;
  return nok;
}


floatt fun(floatt a){ return a;}
void test();

char outfilex[30], outfiley[30];

void filefields(unsigned n)
{ int i;
  FILE *f;
  f=fopen(outfilex,"a");
  fprintf(f,"%e %e",n*dt,X[1]);
  for(i=2; i<=N; i++) fprintf(f," %e",X[i]);
  fprintf(f,"\n");
  fclose(f);
  f=fopen(outfiley,"a");
  fprintf(f,"%e %e",n*dt,Y[1]);
```

```
  for(i=2; i<=N; i++) fprintf(f," %e",Y[i]);
  fprintf(f,"\n");
  fclose(f);
}

int main(int argc, char *argv[])
{ unsigned maxnit=300;
  unsigned lesstimes=1;
  unsigned seed=1;
  boolean writenumbers=False;
  char outfile[20]="self.out";
  if (argc>1)
     { if(argv[1][0]=='h')
         { printf("Self organizing reactional systems -- January 1997 \n"
                  "Michiel Meeuwissen \n\n"
                  " use: %s <options> \n\n"
                  "with options: \n"
                  " N??? : size of grid (%u) \n"
                  " n??? : number of iterations (%u)\n"
                  " t??? : timestep dt (%f)\n"
                  " A??? : value of A (%f)\n"
                  " B??? : value of B (%f)\n"
                  " X??? : initial value of X (%f)\n"
                  " Y??? : initial value of Y (%f)\n"
                  " S?   : initial function number of X and Y (%u)(1-4)\n"
                  "   F??: (S1) frequency of sinusoidal pertubation (%f) \n"
                  "   s? : (S2) seed of random pertubation (%u)\n"
                  " DX???: diffusion coeff. of X (%f)\n"
                  " DY???: diffusion coeff. of Y (%f)\n"
                  " b?   : boundary conditions (%u) (1:'=0 2:eq 3:oth))\n"
                  " M??? : write ??? times less points (%u)\n"
                  " f????: output file-name (%s)\n"
                  "\n\n"
                  ,argv[0],N,maxnit,dt,A,B,initX,initY,startfun,freq,seed,DX,DY,boundary,lesstimes,outfile);
           exit(1);
         }
       for (int arg=1; arg<argc; arg++)
           switch(argv[arg][0])
           { case 'N' : sscanf(argv[arg]+1,"%u",&N); dx=1/N; break;
             case 'n' : sscanf(argv[arg]+1,"%u",&maxnit); break;
             case 't' : sscanf(argv[arg]+1,"%f",&dt); break;
             case 'A' : sscanf(argv[arg]+1,"%f",&A); break;
             case 'B' : sscanf(argv[arg]+1,"%f",&B); break;
             case 'X' : sscanf(argv[arg]+1,"%f",&initX); break;
             case 'Y' : sscanf(argv[arg]+1,"%f",&initY); break;
             case 'S' : sscanf(argv[arg]+1,"%u",&startfun);break;
             case 'D' : switch(argv[arg][1])
                        { case 'X': sscanf(argv[arg]+2,"%f",&DX); break;
                          case 'Y': sscanf(argv[arg]+2,"%f",&DY); break;
                          default : printf("uncorrect use of option D\n");
                        }
                        break;
             case 'f' : sscanf(argv[arg]+1,"%20s",&outfile); break;
             case 'M' : sscanf(argv[arg]+1,"%u",&lesstimes); break;
             case 'w' : writenumbers=True; break;
             case 'b' : sscanf(argv[arg]+1,"%u",&boundary); break;
             case 's' : sscanf(argv[arg]+1,"%u",&seed); break;
             case 'F' : sscanf(argv[arg]+1,"%f",&freq); break;
             default  : printf("unknown option '%c'\n",argv[arg][0]);
           }
     }

strcpy(outfilex,outfile);
strcpy(outfiley,outfile);
strcat(outfilex,".x");
strcat(outfiley,".y");
srand(seed);

resizeglobals();
setglobals();


/*
writefields(4,aX,bX,aY,bY);
itexpl();
*/
  FILE *outf;
  outf = fopen(outfilex,"w");
  fprintf(outf,"# N:%u n:%u dt:%f A:%f B:%f \n# X0:%f Y0:%f  DX:%f DY:%f b:%u\n",
                  N, maxnit, dt, A,   B,   initX, initY, DX, DY,boundary);
  fclose(outf);
  outf = fopen(outfiley,"w");
  fprintf(outf,"# N:%u n:%u dt:%f A:%f B:%f \n# X0:%f Y0:%f  DX:%f DY:%f b:%u\n",
                  N, maxnit, dt, A,   B,   initX, initY, DX, DY,boundary);
  fclose(outf);
  filefields(0);
  for(int j=1; j<maxnit; j++)
   { if (writenumbers) printf("%i\r",j);
     itexpl();
     if (j%lesstimes==0) filefields(j);
     if (testXY()) { printf("Going out of hand (it.: %i), aborting calculation...\n",j); break; }
```

```
   }
//test();
}

void test()
{ unsigned N=4;
  field A(N), B(N), C(N), R(N),U(N);
  A.set(4,-1.,2.,4.,-1.);
  B.set(4,1.,1.,-1.,1.);
  R.set(4,0.,2.,3.,0.);
  writefields(3,A,B,R);
  toggle(A,B);
  writefields(4,A,B,C,R);
  tridag(A,B,A,R,U);  U.write();
}
//
```

C Programma's voor het maken van de plaatjes

Het programma uit het vorige hoofstuk maakt geen plaatjes maar schrijft zijn resultaten slechts naar een file. Met 'gnuplot' kan men eenvoudig kijken wat er met de concentratie op een bepaalde plaats gebeurt als functie van de tijd. Dit kan selfs tijdens het rekenen.

```
set xlabel "t"
set ylabel "concentrations"
set title  "hm"
plot "self.out.x" using 1:20 with lines, "self.out.y" using 1:20 with lines
```

En met MATHEMATICA kan men dan eventueel 'filmjes' maken van de concentratie als functie van de tijd. Dat ging als volgt:

```
(*

*)

If[!NumberQ[reedsanim],
<< Graphics`Animation`;
Print["(s) Graphics Animation is ingeladen"];
reedsanim=1;
];

self = "self.out";

gegevens[naam_]:=
Module[
{file,i,data
},
filenaam = naam;
Print["(s) inlezen  van 'data' uit " <> filenaam <> ".x en -.y ..."];
file = OpenRead[filenaam <> ".x"];
info = Read[file, {String,String}];
info = info[[1]] <> "\n" <> info[[2]];
dataX = ReadList[file, Number, RecordLists->True];
Close[file];
maxt=Length[dataX];
maxx=Length[dataX[[1]]]-1;
dx = 1/(maxx-1);
file = OpenRead[filenaam <> ".y"];
Read[file,{String,String}];
dataY = ReadList[file, Number, RecordLists->True];
Close[file];
dt= dataX[[2,1]] - dataX[[1,1]];
For[i=1, i<=Length[dataX], i++,
    dataX[[i]]=Delete[dataX[[i]],1];
    dataY[[i]]=Delete[dataY[[i]],1];
   ];
maxconc=Max[{dataX,dataY}];
For[i=1, i<=Length[dataX], i++,
    hulp=Table[{(x-1)*dx,dataX[[i,x]]},{x,1,maxx}];
    dataX[[i]]=Interpolation[hulp];
    hulp=Table[{(x-1)*dx,dataY[[i,x]]},{x,1,maxx}];
    dataY[[i]]=Interpolation[hulp];
   ];
Print["(s) dataX en dataY zijn gemaakt en de variable `filenaam'=" <> filenaam];
] (* gegevens *)

red = RGBColor[1,0,0];
green = RGBColor[0,1,0];
blue  = RGBColor[0,0,1];
black = RGBColor[0,0,0];
```

```
tekendata[i_]:=
Plot[{dataX[[i]][x],dataY[[i]][x]},{x,0,1},
              PlotLabel -> "t=" <> ToString[(i-1)*dt],
              PlotStyle -> {red,green},
              PlotRange -> {{0,1},{0,maxconc}},
              AxesLabel -> {"x","concentration"}
        ]


teken:=Animate[tekendata[Round[i]],{i,1,maxt},
       Frames -> 40
              ];

tekent[x_]:=
Module[{},
xt=Table[{dt*(t-1),dataX[[Round[t]]][x]},{t,1,maxt}];
yt=Table[{dt*(t-1),dataY[[Round[t]]][x]},{t,1,maxt}];
xt=Interpolation[xt];
yt=Interpolation[yt];
            Plot[{xt[t],yt[t]},{t,0,dt*(maxt-1)},
                PlotStyle -> {red,green},
                AxesLabel -> {"t", "concentration"},
                PlotLabel -> "x:" <> ToString[x] <> info
                ]
]

gegevensteken:=Module[{},gegevens[self]; teken;]

(*


*)
```

Referenties

[1] Koonin Hoofstuk 7

[2] Nicolis and Prigogine – Self Organization in Nonequilibrium systems – Hoofdstuk 7

[3] Press, et al. – Numerical Recipes in C, second edition

[4] Ammeraal, L. – C++, 3e druk