13 Sep 2015

DISEASE PROPERTIES AFFECT ON DEATH RATES
FYTN03

Johan Book

Department of Astronomy and Theoretical Physics, Lund University

Project done together with Ola Olsson

Contents

1 Introduction

This study aims to investigate how different properties of a disease affect the death rate in the population of a simulated network of cities. Studied parameters are mortality, days of incubation, transmission rate and average days of recovery.

A model extending the SIR model is used in order to simulate a disease outbreak in a procedurally generated network of cities.

2 Theory

2.1 Numerical methods

Let $f(\mathbf{y}_n, t_n)$ denote the derivative of discretized a function $\mathbf{y}_n$ at a time t_n. The easiest approach is the Euler method as presented below. This method has a global error of the order $\mathcal{O}(h^1)$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + hf(\mathbf{y}_n, t_n)$$

One set of slightly more complicated methods are the fourth order Runge-Kutta methods. They are among the most commonly used when solving systems of first order differential equations. They have a global error of the order $\mathcal{O}(h^4)$. Define the following coefficients.

$$\begin{aligned}
\mathbf{k}_1 &= f(\mathbf{y}_n, t_n) \\
\mathbf{k}_2 &= f(\mathbf{y}_n + \tfrac{h}{2}\mathbf{k}_1,\, t_n + \tfrac{h}{2}) \\
\mathbf{k}_3 &= f(\mathbf{y}_n + \tfrac{h}{2}\mathbf{k}_2,\, t_n + \tfrac{h}{2}) \\
\mathbf{k}_4 &= f(\mathbf{y}_n + h\mathbf{k}_3,\, t_n + h)
\end{aligned}$$

One choice of weights in order to achieve $\mathbf{y}_{n+1}$ is the following.

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \tfrac{h}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

2.2 Error analysis

Let f be a function and $g(h)$ be an approximation such that $|f - g(h)|$ decreases as h increases. Define the a global error as $\mathcal{O}(h^n)$ for some integer n. Study the following system of equations.

$$\begin{aligned}
g(h) &= f + \mathcal{O}(2^n h^n) = f + 2^n \mathcal{O}(h) \\
g(2h) &= f + \mathcal{O}(h^n)
\end{aligned}$$

The statement above can be made due to the property $\mathcal{O}(ah) = a\mathcal{O}(h)$.

$$g(2h) - g(h) = 2^n\mathcal{O}(h) - \mathcal{O}(h^n) \Rightarrow \mathcal{O}(h) = \frac{g(2h) - g(h)}{2^n - 1} \tag{1}$$

Thus one has an efficient way of estimating the error. When using e.g. a fourth order Runge-Kutta method, as presented in the former section, the denominator becomes 15.

3 Methods

3.1 Model

3.1.1 Extension to SIR

Let extend the SIR-model presented in [1] to include death and incubation. As in SIR there one group of susceptibles, S, capable of catching the disease and a group of recovered, R, who are immune to the strain. The infected are divided into two groups, I_1 and I_2; one which yet experience no symptoms and one which does. One new group is the dead, D, which are the ones who decease due to the disease. The flowchart below illustrate how these groups relate to each other.

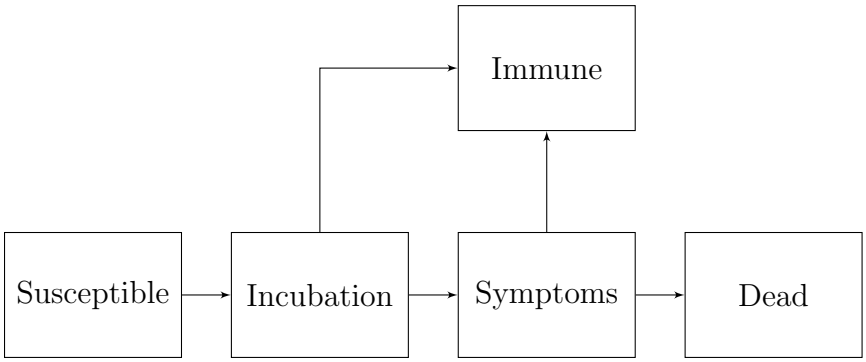


Figure 1: A chart of how the different groups relate to each other.

Due to the addition introduces the derivatives presented in the SIR model need to be revised. Define the the mortality of the disease, $\xi \in [0, 1]$, the incubation time - $\vartheta \in (0, \infty)$ and whether quarantine is utilized, $\Upsilon \in \{0, 1\}$. Assume that those experiencing symptoms travel to a less extent and can thus be ignored.

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\beta S \frac{I_1 + \Upsilon I_2}{N} - \gamma S + \Upsilon \sum_{m \neq n} [\omega_{n \leftarrow m} S_m - \omega_{n \to m} S_n] \tag{2}$$

$$\frac{\mathrm{d}I_1}{\mathrm{d}t} = \beta S \frac{I_1 + \Upsilon I_2}{N} - (\alpha + \frac{1}{\vartheta}) I_1 + \Upsilon \sum_{m \neq n} [\omega_{n \leftarrow m} I_{1m} - \omega_{n \to m} I_{1n}] \tag{3}$$

$$\frac{\mathrm{d}I_2}{\mathrm{d}t} = -\xi I_2 - \alpha I_2 + \frac{1}{\vartheta} I_1 \tag{4}$$

$$\frac{\mathrm{d}D}{\mathrm{d}t} = \xi I_2 \tag{5}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \gamma S + \alpha (I_1 + I_2) + \Upsilon \sum_{m \neq n} [\omega_{n \leftarrow m} R_m - \omega_{n \to m} R_n] \tag{6}$$

One can confirm that the sum of the derivatives is zero as it should. One weak assumption used in this model is that those who experience symptoms meet as many people as those who are not experience any symptoms.

3.1.2 Quarantine

One efficient way of minimizing the spread of a disease is the use of quarantine. Assume when a city is set in quarantine all connections to and from it is shut down and visibly infected are screened off from the rest of the city population. In reality this is a most expensive and serious action and most only be utilized if really required. In order to create a measure of whether quarantine should be utilized or not define a threshold $\nu \in (0, 1)$.

$$\Upsilon = H\left[\max\left(\frac{I_2}{N} - \nu, \frac{1}{N\nu}\frac{\mathrm{d}I_2}{\mathrm{d}t} - \nu\right)\right]$$

H is the Heaviside step function. An more sensitive version is the following.

$$\Upsilon = H\left(\frac{I_2}{N} + \frac{1}{N\nu}\frac{\mathrm{d}I_2}{\mathrm{d}t} - \nu\right)$$

Neither of these methods for determining quarantine was implemented.

3.1.3 Time periods for model validity

The simulation is meant to study the disease during a few months, hence making the population changes as births and natural deaths negligible. Therefore this model is most reliable when studying shorter outbreaks, preferably less than a year.

3.2 Numerical method and error estimation

The chosen numerical method is the Runge-Kutta method presented 2.1. It does give a great accuracy consider and is relatively easy to implement. The Euler method was also considered due to its simplicity and fast computation time but was rejected due to the minimal error of the Runge-Kutta method. The implementation of this is presented in the appendix. The main part of the the calculations are carried out in A.4.

The error estimation uses the method proposed in 2.2. The simulation is carried out twice, once with a double step and once with a normal step allowing for use of said method.

3.3 Network generation

In order to create a network of cities a position and size of population was randomly drawn given two conditions. The point was picked from a uniform distribution and the population from a narrow Gauss distribution.

1. The distance between the picked point and all other cities must be above a certain threshold.

2. The population must belong to a defined range with a lower and upper bound.

3.3.1 Road generation

Let define a quantity labelled travel-rate between two cities as the following. The constant C was chosen to 1/5000.

$$\lambda(a, b) = C \frac{\text{Population } a + \text{Population } b}{\text{Distance}(a, b)}$$

Roads were drawn according to two rules.

1. Draw for each city a a road to another city b for which $\lambda(a, b)$ is minimal.

2. For any cities a and b draw a road if $\lambda(a, b) \geq 1$.

Along each road a travel-rate $\omega(a, b)$ was calculated for each time step using the formula below. Let define a quantity labelled travel-rate between two cities as the following. N is the total population throughout the network and C is a constant chosen to 5/1000.

$$\omega(a, b) = \frac{C}{N} \frac{\text{Population } a}{\text{Distance}(a, b)}$$

$\omega(a, b)$ is used as $\omega_{a \leftarrow b}$ in eq. 2, 3 and 6.

3.3.2 Name generation*

In order to randomly generate city names three groups of strings were utilized; one with prefixes such as "North", one with suffixes such as "City", one with the first part of the name as as "Lon" from "London" and finally a group with second part of the word e.g. "don". A name was generated according the following rules.

1. Randomly pick either a prefix, a suffix or do neither.

2. Pick a first part of the name. If no prefix or suffix was used this part may randomly be replaced by a prefix.

3. Pick a second part of the name.

4. Put it all together

4 Results

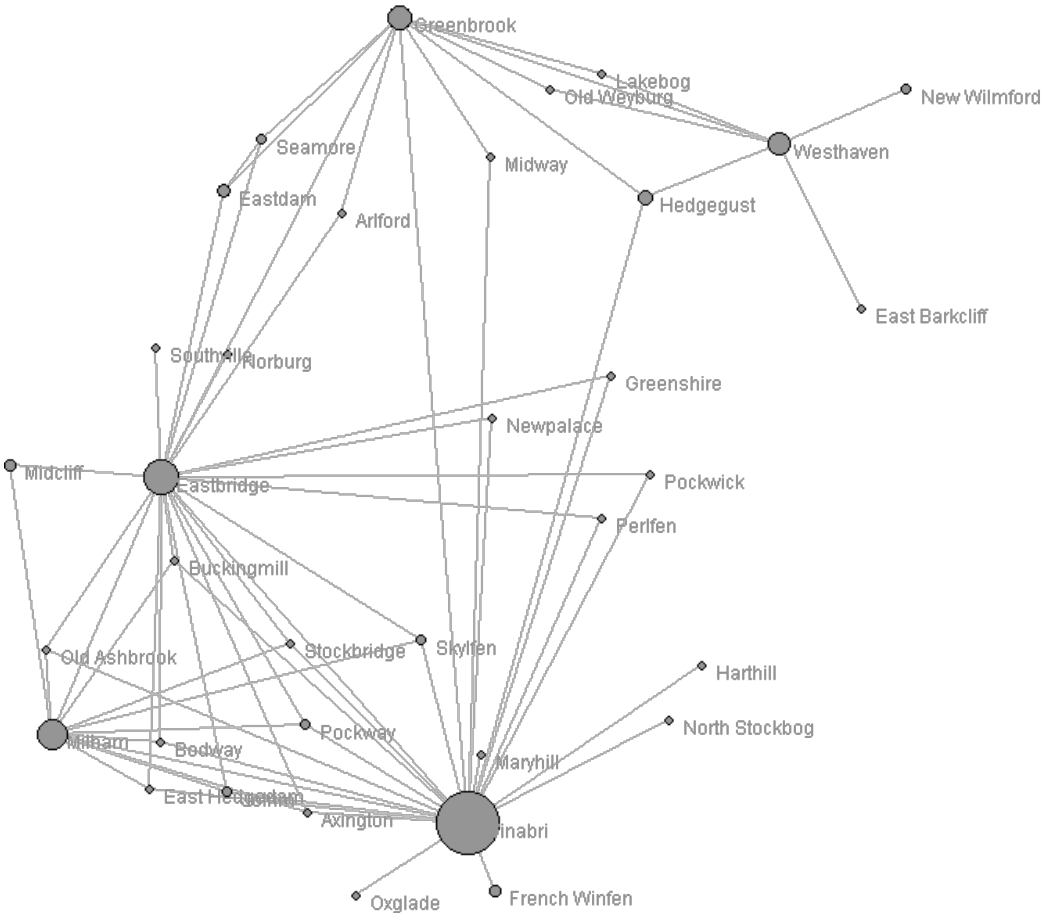


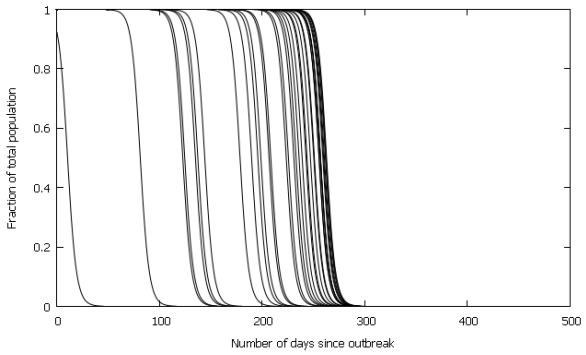
Figure 2: A map over the studied city network.

One randomly generated network consisting of 35 cities was chosen due to its interesting connection properties. The disease outbreak started in city of New Wilmford. The following parameters was used:

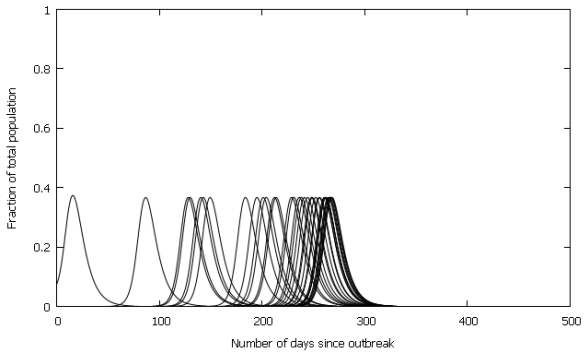
Parameter	Value
α	0.01
β	0.25
ϑ	10
ξ	0.01
Time period	50 days
Time interval	1 day

Table 1: Parameters used in 4.1.

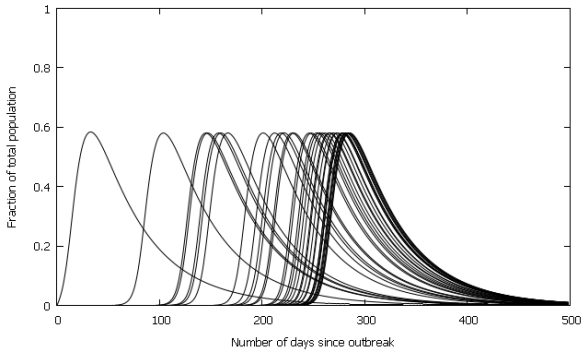
4.1 Network results



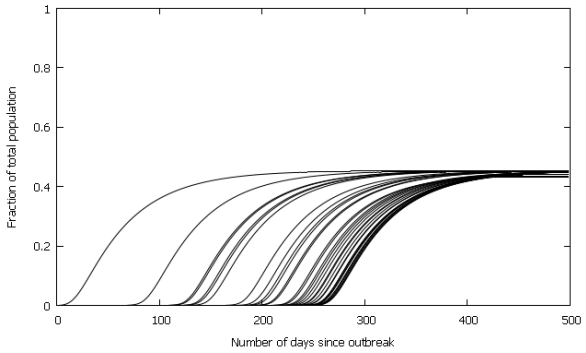
(a) Susceptible.



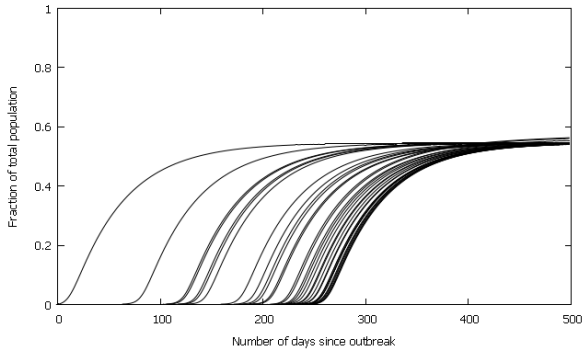
(b) Incubation.



(c) Symptoms.



(d) Dead.

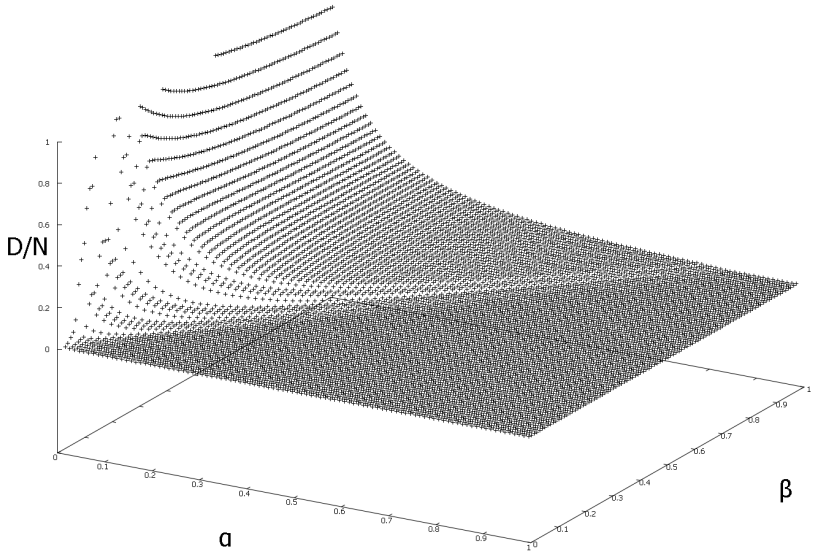


(e) Recovered.

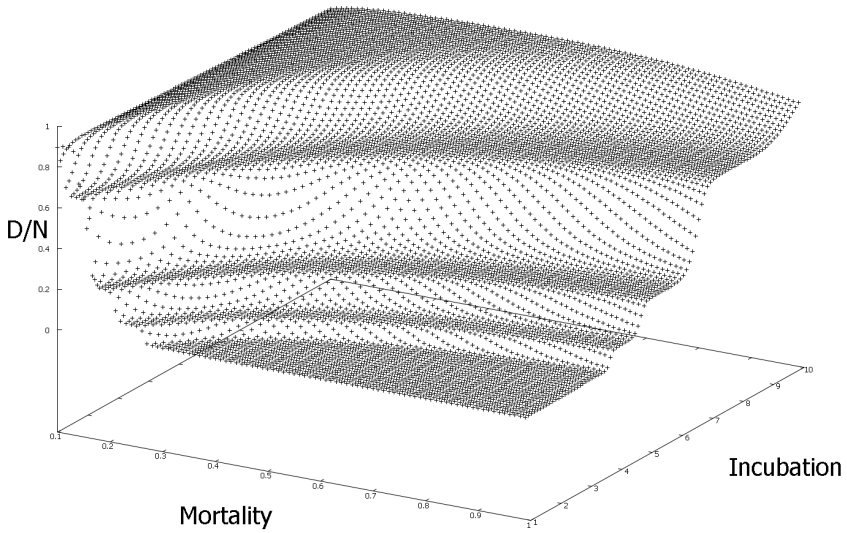
Figure 3: How each group relates to total population as a fraction. One line correpsonds to a group in one city.

How each group changed at every time step was studied and is presented in the figures above. The maximum error in each fraction in the figures above is of the order 10^{-8}, calculated using eq. 1.

4.2 Landscapes



(a) Fraction as a function of α and β. $\xi = 0.5$ and $\vartheta = 15$.



(b) Fraction as a function of mortality ξ and incubation ϑ. $\alpha = 0.01$ and $\beta = 0.25$.

Figure 4: Fraction dead of total population over the network as function of different simulation parameters.

5 Analysis

The total death rate seem to approach 45 % of the total population while recovered approach 55 %. This is not surprising since these are the only remaining groups hence should reach stability given enough time. Since all populations share the same properties the point of stability is the same in every city together with the same curve characteristics. Said points should be possible to calculate analytically given the model parameters. However no such relation have been found.

5.1 Landscape for α and β

The landscape presented in fig. 4a is flat for $\alpha > \beta$ which was to be expected since if the rate of recovery, α is greater than the rate of transmission, β, the disease will wear away. However, if $\beta > \alpha$ does $\frac{D}{N}$ seem to increase exponentially as α decreases. This implies that in this model does the transmission rate have little effect as long as the recovery rate is low. As $\alpha \to 0$ the disease will result in a mass extinction.

5.2 Landscape for ξ and ϑ

The landscape in fig. 4b implies that for long incubation times a disease will reach a certain threshold discussed in a former paragraph - given enough time. For lower incubation times there is a ladder-type of behaviour where those who have lower mortality in general result in more deaths. This is due to the weak assumption made in the model that those who experience symptoms socialize to same degree as those who do not experience any symptoms. This affect the model by favouring parameters where those who experience symptoms are maximized. Redoing the model without this assumption could hence further improve the results.

The equi-surfaces in the ladder are points where $\frac{\partial D}{\partial \xi} = \frac{\partial D}{\partial \vartheta} = 0$ and are probable regions for actual diseases since a disease where any of these derivatives are non-zero are more likely to change.

5.3 Conclusion

This model suggests that diseases with a long incubation time are the most hazardous - which does seem plausible. The fact that the model favours low mortality is less realistic and probably due to a weak assumption in the model.

A Code for numerical approximation

A.1 Differentiable

```
package solver;

import settings.Settings;

public interface Differentiable {
   public double[] differentiate(double[] y, double[] x, double t, final
       Settings settings);
}
```

A.2 ExtendedSIR

```
package solver; import settings.Settings;

public class ExtendedSIR implements Differentiable {

   @Override
   public double[] differentiate(double[] y, double[] x, double t, final
       Settings settings) {
      double[] dy = new double[y.length];

      for (int i = 0; i < settings.number_of_cities; i++) {
         // Calculate travel
         double[] travelsum = new double[settings.number_of_groups];
         if (!settings.network.cities[i].quarantine)
            for (int j = 0; j < settings.number_of_cities; j++) {
               if (i == j || settings.network.cities[j].quarantine)
                  continue;
               for (int group = 0; group < settings.number_of_groups; group++)
                  travelsum[group] -= x[i + settings.number_of_cities * j]
                        * (y[i * settings.number_of_groups + group] - y[j
                              * settings.number_of_groups + group]);
            }
         int k = i * settings.number_of_groups;

         // Get total population
         double N = y[k + 0] + y[k + 1] + y[k + 2] + y[k + 4];
         // Susceptible
         dy[k + 0] = -settings.beta * y[k + 0] * (y[k + 1] + y[k + 2]) / N
               + travelsum[0];
         // Incubation
         dy[k + 1] = settings.beta * y[k + 0] * (y[k + 1] + y[k + 2]) / N
               - settings.alpha * y[k + 1] - y[k + 1] / settings.incubation +
                   travelsum[1];
         // Sick
         dy[k + 2] = y[k + 1] / settings.incubation - y[k + 2]
               * settings.deathrate - settings.alpha * y[k + 2];
         // Dead
         dy[k + 3] = y[k + 2] * settings.deathrate;
         // Recovered
         dy[k + 4] = settings.alpha * (y[k + 1] + y[k + 2]) + travelsum[4];
      }
      return dy;
   }
}
```

A.3 Solver

```
package solver;

import settings.Settings;

public abstract class Solver {
   public abstract double[] solve(Differentiable function, double[] y, double[]
       x, double t, double h, final Settings settings);

   public static double[] add(double[] a, double[] b) {
      if (a.length != b.length)
         return null;
      double[] c = new double[a.length];
      for (int i = 0; i < a.length; i++)
         c[i] = a[i] + b[i];
      return c;
   }

   public static double[] multiply(double[] a, double t) {
      double[] c = new double[a.length];
      for (int i = 0; i < a.length; i++)
         c[i] = a[i] * t;
      return c;
   }

   public static double[] sum(double[]... a) {
      double[] c = new double[a[0].length];
      for (double[] x : a)
         c = add(c, x);
      return c;
   }

   public static double total(double[] a) {
      double sum = 0;
      for (double x : a)
         sum += x;
      return sum;
   }
}
```

A.4 RungeKutta4

```
package solver;

import settings.Settings;

public class RungeKutta4 extends Solver {

   @Override
   public double[] solve(Differentiable function, double[] y, double[] x,
         double t, double h, final Settings settings) {
      double[] k1 = function.differentiate(y, x, t, settings);
      double[] k2 = function.differentiate(add(y, multiply(k1, 0.5 * h)), x,
            t + 0.5 * h, settings);
      double[] k3 = function.differentiate(add(y, multiply(k2, 0.5 * h)), x,
            t + 0.5 * h, settings);
      double[] k4 = function.differentiate(add(y, multiply(k3, h)), x, t + h,
            settings);
      double[] sum = multiply(sum(k1, multiply(k2, 2), multiply(k3, 2), k4),
            h / 6);
      return add(y, sum);
   }

}
```

References

[1] Lund University, *ODE, Lecture 2, FYTN03 Computational Physics.*