A Solution to ARGESIM Benchmark C17 'SIR-type Epidemic' using Numerical Programming Features in MAPLE

Štefan Emrich, Hannes Glavanovits, Tanaz Khorzad,
Vienna University of Technology, Austria; *Stefan.Emrich@tuwien.ac.at*

Simulator: *MAPLE* has primarily been designed as computer algebra system for analytical/symbolical computation. Recent versions support also numerical vector and matrix manipulations on a high level, so that also numerical tasks of any kind can solved. Among several numerical algorithms, MAPLE offers solvers for ODEs and DAEs in convenient way (additionally with symbolic solutions, e.g. by series). Usually, users work in a Java-based GUI, with command window, display window, etc., where commands are put in straightforward. Programming in terms of complex programs - as used in this solution - is possible although not very convenient.

Modelling. Three modelling approaches for SIR-type epidemics have to be implemented: ODEs, difference equations (DEs) and cellular automata (CA). ODE modelling and solving is a standard task of MAPLE: ODEs are defined, then the solver is called:

```
a:=0.2: r:=0.6/10^4:
ics:=K(0)=100, S(0)=16000, R(0)=0 :
ode:=diff(S(t),t)=-r*S(t)*I(t),
     diff(I(t),t)=r*S(t)*I(t)-a*I(t),
     diff(R(t),t)=a*I(t):
solution:=dsolve({ode,ics},numeric);
```

Difference equations must be implemented in nested loops. The Euler discretisation with unit stepsize for the ODEs makes use of arrays and loops:

```
for i from 1 to 50 do
 Sus[i]  := Sus[i-1]*(1-r)^(Inf[i-1]/16000);
 Inf[i]  := Inf[i-1]+Sus[i-1]*
      (1-(1-r)^(Inf[i-1]/16000))-a*Inf[i-1];
 Rec[i]  := Rec[i-1]+a*Inf[i-1];
end do:
```

A much more elaborate task is the implementation of a cellular automata model. As MAPLE is not capable of handling subroutines in terms of external functions/programs all the routines need to be written in a single file in nested loops. In principle, the CA is implemented by arrays representing the grid of cells of the automaton, which are updated in unit steps. Two reasons suggest to structure the update of the cells: first, the one-file implementation in MAPLE needs to be structured for better coding, debugging, and reading; and second, codes for CA update become standardised using evolution operators like *update*, *propagation*, *transition*, etc. Consequently the MAPLE implementation follows this suggestion: manipulation of the CA is realised by evolution functions like *SetParticles*, *ParticleMovementHPP*, *InfectionTotal*, etc. These functions are then called repeatedly within a loop and so resemble the CA. The implementation has been kept very flexible to allow changes in the general structure (automata dimension, lattice structure, etc.)

The main loop for the cell state update moves the particles = individuals (`MovementParticle`), checks for collision with or without infection (`Collision`), checks for recovery (`Recovery`), and summarises the new amount of susceptible, infected and recovered particles (`RecollectQuantity`) for comparison with ODE and DE solutions :

```
CellularAutomaton:=proc(quant)
  local counter:
  counter:=0:
    while counter<quant do
      MovementParticleFHP():
      CollisionFHP():
      Recovery():
      RecollectQuantity();
      counter:=counter+1:
    end do:
```

Appendices characterise the special structure of the lattice gas cellular automata used (HHP: Hardy - de Pazzis - Pomeau automaton; FHP: Frisch - Hasslacher - Pomeau automaton). As example, the FHP movement on a square grid is implemented by:

```
MovementParticleFHP:=proc()
global Cells, Cellcopy, ... : local i,j,k:
for k from 1 by 1 to ParticlePerCell do
 for i from 1 by 1 to Cell_width do
  for j from 1 by 1 to Cell_length do
   Cellcopy[MoveFHP[k](i,j,k)] :=
                Cells[i,j,k] :
  end do:  end do: end do:
```

A - Task: CA and ODE Simulation. MAPLE works on basis of procedures: for ODE solution, first the ODE is defined and parametrised (see before); then a solver procedure is set up, to be used for calculating the solution `solution` at time instants `solution(i/100000000)` (results in Figure 1):

```
solution(0);
for i from 1 to 10 do
  solution(i/100000000): end:
odeplot(lsg,[[t,S(t)],[t,K(t)],
             [t,R(t)]],0..100);
```

For CA simulation, first the grids with the cell states are initialised with random distribution of the particles. Cellular automaton simulation is started by calling `Cellular_Automaton:=proc(quant)` for the desired number of updates (`quant`), using a specified CA type.

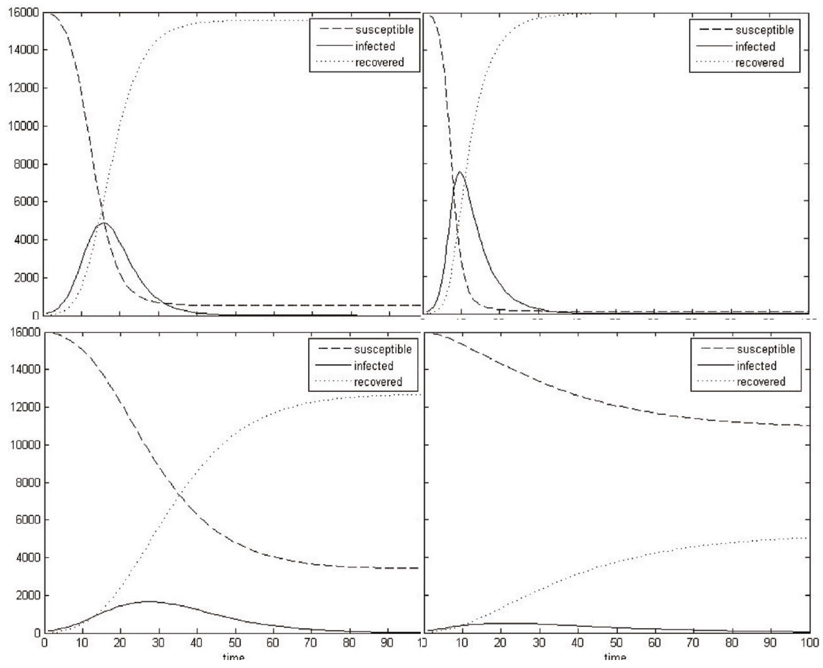

Figure 1: Results of different SIR models - a) ODE solution, b) FHP-CA solution, c) HPP-CA solution, d) random HPP-CA.

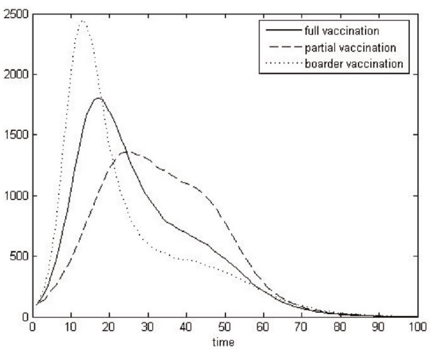

Figure 2: Infected for different vaccination strategies, FHP-CA.

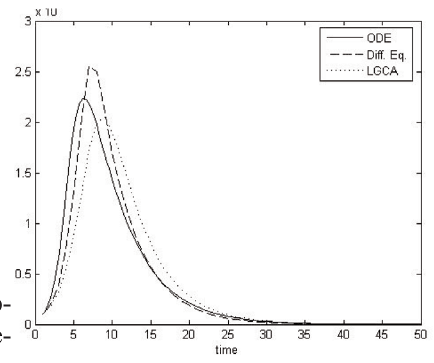

Figure 3: Infected for different models: ODEs, DEs, 'uniform' FHP-CA.

The types investigated are FHP-CA, HPP-CA, and HPP-CA with random deflection. The results (Figure 1) show qualitatively similar behaviour, but the CA dynamics is significantly slower.

B - **Task: Vaccination Strategies in CAs.** For the FHP-CA model, different vaccination strategies are modelled by different initial distribution of recovered individuals (particles) on the cell grid, because recovered people behave like vaccinated - they cannot get infected. An initial distribution of recovered individuals (4.000) can be implemented as follows:

```
CellNumber:=proc(status,number,xUpL,...)
while j<number do
 row:=Generate(integer(distribution=
  uniform[ yUpL-1, ylowR],projection=ceil));
 column:=Generate(integer(distribution=
  uniform[ xUpL-1, xLowR],projection=ceil));
end do:end proc:
```

Simulation results (Figure 3; infected individuals) for the different vaccination strategies show, that the solutions do not differ significantly. But interestingly, partial area vaccination results in less infected individuals than full area vaccination (all with 4.000 vaccinated).

C - **Task: ODE vs. CA Solutions.** ODE solution and DE solution are references for the summed up dynamics of suscepted, infected and recovered individuals of CA models, in case of spatial equal distribution of all individuals in each time step. For comparison, the ODE solution is calculated as before, and the DE solution is calculated by the simple loop shown in Task **M**.

In FHP-CA update after movement, collision, and recovery, all individuals must be equally distributed on the grid, as given in the code snippet below. Results in Figure 3 show indeed a good coincidence of all three solutions.

```
MovementParticleFHP(): CollisionFHP():..
R:=Eval(3); I:=Eval(2); S:=Eval(1);
NumberRandom(3,R,1,1,C_width,C_length):
NumberRandom(2,I,1,1,C_width,C_length):...
RandomSetParticles():
```

R**ésumé:** While ODE solution is a standard task for MAPLE 10 in this benchmark solution, modelling of a CA and programming of a CA update algorithm is a nontrivial task. The chosen implementation works with arrays for the CAs, which are updated by evolution functions programmed as MAPLE procedure. The implementation follows suggestions for standardised evolution operators for CA modelling and requires deep knowledge of MAPLE programming.

Corresponding Author: Štefan Emrich,
H. Glavanovits, T. Khorzad, *Stefan.Emrich@tuwien.ac.at*
Vienna University of Technology, Austria;
Inst. f. Analysis and Scientific Computing,
Wiedner Hauptstrasse 8-10, 1040 Vienna, Austria

Reveived: September 20, 2006
Revised: October 25, 2006; November 16, 2006
Accepted: December 10, 2006