An. şt. Univ. „Al. I. Cuza", Iaşi
Informaică, 35 (1989)

PROGRAM DESIGN FOR THE NUMERICAL SOLUTION OF AN OPTIMAL CONTROL PROBLEM

BY

VIOREL ARNĂUTU

1. Introduction. The design of a program for the numerical solution of an optimal control problem is faced. The control problem, which is considered in connection with a class of man-environment epidemic systems in [1], is presented in §2 together with a numerical solution. The implementation of the algorithm, starting from its mathematical foundations, is discussed in §3. A simple Pascal-like language is used to describe different modules of the program. The numerical behavior of the program is given in §4.

2. Statement of the control problem and the descent algorithm. An important class of epidemics is that of man-environment type; for such a class of epidemic systems the environment sustains the epidemic itself by a positive feedback driven by the infectious agent which is produced by the infective human population. The habitat is represented by a bounded and open domain $\Omega \subset \mathbb{R}^2$ with a sufficiently smooth boundary $\partial\Omega = \Gamma_1 \cup \Gamma_2$, where Γ_1 is the seashore and Γ_2 is the land boundary. Γ_1 and Γ_2 have disjoint interiors (see Fig. 1).

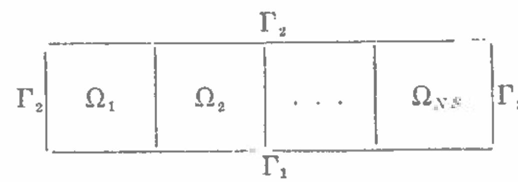

Fig. 1

We denote by $u_1(t, x)$ the spatial density of the infectious agent at moment $t \in [0, T]$ and at point $x \in \Omega$; $u_2(t, x)$ is the number of infectious individuals. A realistic model must take into account the fact that the sewage produced by the human population is sent untreated into the sea and from there the infectious agent goes back to the habitat via the seashore through some diffusion mechanism related to the habits of the local population. The corresponding state system is given by (see also [1])

$$\frac{\partial u_1}{\partial t}(x, t) - \Delta u_1(x, t) + a_{11} u_1(x, t) = 0 \text{ in } Q = \Omega \times (0, T), \tag{2.1}$$

$$\frac{\partial u_2}{\partial t}(x, t) + a_{22} u_2(x, t) - g(u_1(x, t)) = 0 \text{ in } Q, \tag{2.2}$$

(2.3) $u_1(x, O) = u_1^0(x), \ u_2(x, O) = u_2^0(x)$ for $x \in \Omega$,

(2.4) $$\frac{\partial u_1}{\partial \nu}(\sigma, t) + \alpha u_1(\sigma, t) = K^* u_2 = \int_\Omega K(x, \sigma, t) u_2(x, t) dx$$

for $(\sigma, t) \in \Sigma_1 = \Gamma_1 \times (O, T)$,

(2.5) $$\frac{\partial u_1}{\partial \nu}(\sigma, t) = 0 \text{ for } (\sigma, t) \in \Sigma_2 = \Gamma_2 \times (O, T).$$

— Equation (2.1) describes the diffusion of the infectious agent into the habitat; we have $a_{11} > 0$ and a_{11}^{-1} is the mean lifetime of the agent in the environment.

— Equation (2.2) describes the evolution of the human infective; we have $a_{22} > 0$ and a_{22}^{-1} is the mean infectious period of the human infective; g is the "force of infection" due to the agent. The shape of the function g strongly depends on the kind of epidemics.

— The initial conditions are given by (2.3).

— the "feedback" at the boundary Γ_1 is modelled by a linear integral operator (Equation (2.4)); we take into account the transfer of the infectious agent generated in different parts of the habitat Ω to the seashore. We consider

(2.6) $$\begin{cases} K(x, \sigma, t) = w(t) K_0(x, \sigma), \\ K_0(x, \sigma) = \sum_{i=1}^{NS} a_i \tilde{\chi}_i(x) \chi_i(\sigma). \end{cases}$$

where $w(t)$ — the cost of the sewage treatment-represents the control of the system, a_i are weights, $\tilde{\chi}_i$ is the characteristic function of the subdomain Ω_i and χ_i is the one for $\Gamma_1 \cap \partial\Omega_i$.

— Equations (2.5) states that the boundary Γ_2 is completely insulated.

The aim of the optimal control problem is to reduce the epidemic phenomenon described by Eqns. (2.1)-(2.5) by reducing the boundary feedback at the seashore i.e., by reducing the "strength" of the kernel K. This corresponds to the implementation of a sanitation program by means of a treatment of the sewage before sending it to the sea. Its cost has to be compared with the cost of the epidemic itself. Hence the following optimal control problem is considered

Problem (P) : *Minimize*

$$\Phi(w) = \int_0^T \int_\Omega f(u_2(x, t)) dx dt + \int_0^T h(w(t)) dt + \int_\Omega l(u_2(x, T)) dx$$

for all (u_1, u_2, w) *subject to state system* (2.1) — (2.5).

An existence result and the first order necessary optimality conditions for problem (P) are established in [1]. Gradient type algorithms are derived in order to approximate the optimal solution. Let us consider such a conceptual algorithm ALG :

Step 0 : Let $w^{(1)}$ be arbitrarely choosen in $L^\infty(O, T; R)$;

Set $j := 1$;

Step 1 : Compute $(u_1^{(j)}, u_2^{(j)})$ solving

(2.7) $$\begin{cases} \partial u_1^{(j)}/\partial t - \Delta u_1^{(j)} + a_{11} u_1^{(j)} = O \text{ in } Q, \\ \partial u_2^{(j)}/\partial t + a_{22} u_2^{(j)} - g(u_1^{(j)}) = O \text{ in } Q, \\ u_1^{(j)}(x, O) = u_1^0(x), \quad u_2^{(j)}(x, O) = u_2^0(x) \text{ for } x \in \Omega, \\ \partial u_1^{(j)}/\partial \nu + \alpha u_1^{(j)} = K^* u_2^{(j)} \text{ on } \Sigma_1, \\ \partial u_1^{(j)}/\partial \nu = O \text{ on } \Sigma_2. \end{cases}$$

Step 2 : Compute $(p_1^{(j)}, p_2^{(j)})$ solving

(2.8) $$\begin{cases} \partial p_1^{(j)}/\partial t + \Delta p_1^{(j)} - a_{11} p_1^{(j)} + g'(u_1^{(j)}) p_2^{(j)} = O \text{ in } Q, \\ \partial p_2^{(j)}/\partial t - a_{22} p_2^{(j)} + w^{(j)} \int_{\Gamma_1} K_0(x, \sigma) p_1^{(j)}(\sigma, t) d\sigma = f'(u_2^{(j)}) \text{ in } Q, \\ p_1^{(j)}(x, T) = O, \quad p_2^{(j)}(x, T) = -l'(u_2^{(j)}(x, T)) \text{ for } x \in \Omega, \\ \partial p_1^{(j)}/\partial \nu + \alpha p_1^{(j)} = O \text{ on } \Sigma_1, \\ \partial p_1^{(j)}/\partial \nu = O \text{ on } \Sigma_2. \end{cases}$$

Step 3 : Compute

$$d^{(j)} := \nabla\Phi(w^{(j)}) = h'(w^{(j)}) - \sum_{i=1}^{NS} a_i \int_{\Omega_i} u_2^{(j)} dx \int_{\partial\Omega_i \cap \Gamma_1} p_1^{(j)} d\sigma.$$

Step 4 : IF $\|d^{(j)}\| < \varepsilon$

THEN *Stop*

ELSE Let $w^{(j+1)} := w^{(j)} - \rho_j d^{(j)}$ be such that

$$\Phi(w^{(j+1)}) = \min_{\rho > 0} \Phi(w^{(j)} - \rho d^{(j)});$$

Set $j := j + 1$;

GO TO Step 1.

The convergence of the algorithm is discussed in [1].

It results that $\lim_{j \to \infty} \nabla\Phi(w^j) = 0$ and every (weak) accumulation point w^* of the sequence $\{w^j\}$ satisfies $\nabla\Phi(w^*) = 0$.

3. **Implementation of the algorithm.** The computer implementation of the algorithm is made using the following "heavy" modules

1. STATE — solves the state system (Step 1 in ALG).
2. ADJST — solves the adjoint system (Step 2 in ALG).
3. A "module" to fit the step ρ_j of the gradient method at each interation.

The differential problems are discretized by finite difference schemes which are well suited for equations with positive values. Consider $\Omega = (a, b) \times (c, d)$ and the grids with equidistant knots

$$a = x_1 < x_2 < \ldots < x_{M+1} = b,$$

$$c = y_1 < y_2 < \ldots < y_{N+1} = d,$$

$$O = t_1 < t_2 < \ldots < t_{NT+1} = T.$$

The corresponding arrays are $XKNOT$, $YKNOT$, $TKNOT$ and the grid steps are DX, DY and DT respectively. We introduce tridimensional arrays denoted u for u_1, v for u_2, p for p_1 and q for p_2 and indexed as $u(i, j, k) = u_1(x_i, y_j, t_k)$.

The module STATE. The differential system (2.7) is decoupled as (we drop the iteration index)

$$
(3.1)\qquad \begin{cases} \dfrac{\partial u}{\partial t}(x, y, t) - \Delta u(x, y, t) + a_{11}u(x, y, t) = 0 \text{ in } Q, \\ u(x, y, t_1) = u_1^0(x, y) \text{ for } (x, y) \in \Omega, \\ \dfrac{\partial u}{\partial \nu}(\sigma, t) + \alpha u(\sigma, t) = \displaystyle\iint_{\Omega} K(x, y, \sigma, t)v(x, y, t)dx\,dy \text{ on } \Sigma_1, \\ \dfrac{\partial u}{\partial \nu}(\sigma, t) = 0 \text{ on } \Sigma_2 \end{cases}
$$

and

$$
(3.2)\qquad \begin{cases} \dfrac{dv}{dt}(x, y, t) + a_{22}v(x, y, t) - g(u(x, y, t)) = 0 \text{ in } Q, \\ v(x, y, t_1) = v_2^0(x, y) \text{ for } (x, y) \in \Omega, \end{cases}
$$

where (3.2) is an ODE in t for each $(x, y) \in \Omega$. Since in [1] it is demonstrated that the map $u_2 \to u_1$ is a contraction between appropriate Sobolev spaces, a fixed point method is used to solve numerically (3.1)+(3.2).

Denote $u^{(k)} = u^{(k)}(x, y) = u(x, y, t_k)$. The second equation in (3.1) gives the values of $u^{(1)}$ and the first one is semidiscretized by the standard implicit scheme leading to (3.3)

$$
(3.3)\qquad -(a_{11} + 1/DT)u^{(k+1)} + \Delta u^{(k+1)} = -u^{(k)}/DT \text{ in } \Omega,\ k = 1, 2, \ldots, NT.
$$

For every fixed k this is an elliptic PDE. These equations are solved ascending with respect to time levels using the boundary conditions in (3.1)

$$
(3.4)\qquad \begin{cases} \dfrac{\partial u^{(k+1)}}{\partial x}(a, y) = \dfrac{\partial u^{(k+1)}}{\partial x}(b, y) = 0 \text{ for } y \in (c, d). \\ \dfrac{\partial u^{(k+1)}}{\partial y}(x, c) - \alpha u^{(k+1)}(x, c) = \\ = -w(t_{k+1}) \displaystyle\sum_{i=1}^{NS} a_i \iint_{\Omega_i} v(x, y, t_{k+1})dx\,dy \text{ for } x \in (a, b), \\ \dfrac{\partial u^{(k+1)}}{\partial y}(x, d) = 0 \text{ for } x \in (a, b). \end{cases}
$$

The elliptic systems have been solved with a performant code (see Acknowledgement). We shall denote it by $ELLIPT$. The corresponding boundary values are stored in the vectors BDA, BDB, BDC and BDD respectively. The right-hand side of (3.3) is stored in the bidimensional array RHS and

$ELLIPT$ stores the solution in the bidimensional array SOL. The system (3.2) may be integrated mathematically giving

$$
(3.5)\qquad v^{(k+1)}(x, y) = \exp(-a_{22}t_{k+1})\left(v^{(1)}(x, y) + \int_0^{t_{k+1}} \exp(a_{22}\tau)g(u(x, y, \tau))d\tau\right).
$$

The design of the algorithm is given in a Pascal-like language which is straightforward (we do not insist on the syntax of this „language" since this is not important for our goal). We present first the array initialization which is made in the main program

```
INITIALIZE—ARRAYS-1 ;
Begin
    for i : = 1 to M+1 do
    for j : = 1 to N+1 do
          u(i, j, 1) : = UINIT (xknot (i), yknot (j)) ;
          temp : = VINIT (xknot (i), yknot (j)) ;
          vnew (i, j, 1) : = temp ;
          for k : =1 to NT+1 do
          vold (i, j, k) : = temp
          next k ;
          next j ;
          next i ;
   for j : =1 to N+1 do
       bda (j) : =0 ;
       bdb (j) : =0
       next j ;
   for i : =1 to M+1 do
       bdd (i) : = 0
       next i
End ;
```

The function type routines $UINIT$ and $VINIT$ compute the initial conditions given by u_1^0 and u_2^0 respectively. We give next the design of $STATE$. Only the "main" parameters are included as arguments. Since the computer program was written in FORTRAN many parameters have been transmitted using COMMON blocks. Let us explain that

EPS = required precision in the l^∞-norm for u_2 ;

$MAXITER$ = number of iterations allowed to the fixed point method ;

$KODERR$ = error parameter

=1 — the convergence of the fixed point method is obtained
=0 — otherwise.

```
SUBROUTINE STATE (u, vold, vnew, w, EPS, MAXITER, KODERR)
Begin
koderr : = 0 ;
iter : = 1 ;
while (koderr = 0) and (iter ≤ MAXITER) do
      {compute u(vold) solving (3.1)}
      for i : = 1 to M+1 do
      for j : = 1 to N+1 do
      rhs (i, j) : = -u(i, j, 1)/DT
      next j
      next i ;
      for k : = 1 to NT do
            temp : = RFCT (vold, k+1,...) ;
            for i : =1 to M+1 do
            bdc (i) : =temp
            next i ;
            call ELLIPT (rhs, sol, bda, bdb, bdc, bdd) ;
            for i : =1 to M+1 do
            for j : =1 to N+1 do
            u(i, j, k+1) : =sol (i, j) ;
            rhs (i, j) : = -sol (i, j)/DT
            next j
            next i
 next k ;
 {compute vnew (u) given by (3.5)}
 {and ||vnew-vold||}
 err : =0 ;
 for k : = 2 to NT+1 do
       for i : = 1 to M+1 do
       for j : =1 to N+1 do
       vnew (i, j, k) : = VFCT (xknot(i), yknot(j), tknot(k), vold,...) ;
       temp : =ABS (vnew (i, j, k)-vold (i, j, k)) ;
       if err<temp
       then err : = iemp
       end - if
       next j
       next i
       next k ;
       if err ≤ EPS
       then koderr : =1
       else iter : = iter+1
  end - while
  End {STATE} ;
```

Two function type routines are called :

RFCT — computes the right-hand side of the boundary condition (3.4) for t_{k+1} fixed ; a Simpson bidimensional formula is used.

VFCT — computes v given by (3.5) for fixed (x_i, y_j) by use of a numerical integration formula.

If *STATE* ends with $koderr = 1$ then arrays u and $vnew$ contain the solution of the state system.

The module ADJST. It is similar to *STATE*. The system (2.8) is decoupled as (we use p instead of p_1 and q instead of p_2) :

$$(3.6)\quad \begin{cases} \dfrac{\partial p}{\partial t}(x, y, t)+\Delta p(x, y, t)-a_{11}p(x, y, t)=-g'(u(x, y, t))q(x, y, t) \text{ in } Q, \\ p(x, y, T)=0 \quad \text{for} \quad (x, y)\in\Omega, \\ \dfrac{\partial p}{\partial \nu}(\sigma, t)+\alpha p(\sigma, t)=0 \text{ on } \Sigma_1, \\ \dfrac{\partial p}{\partial \nu}(\sigma, t)=0 \text{ on } \Sigma_2. \end{cases}$$

$$(3.7)\quad \begin{cases} \dfrac{\partial q}{\partial t}(x, y, t)-a_{22}q(x, y, t)=f'(v, (x, y, t))-w(t)\sum\limits_{i=1}^{NS} a_i \displaystyle\int_{x_{1,i}}^{x_{2,i}} p(x, c, t)dx \text{ in } Q, \\ q(x, y, T)=-l'(v(x, y, T)) \text{ for } (x, y)\in\Omega. \end{cases}$$

The system (3.6) is similar to (3.1) .The main difference is that we start with the final condition $p^{NT+1}\equiv 0$ and solve descending with respect to time levels the corresponding elliptic problems given by

$$(3.8)\quad \begin{gathered} \Delta p^{(k)}-(a_{11}+1/DT)p^{(k)}=-p^{(k+1)}/DT-g'(u^{(k)})q^{(k)} \\ \text{in } \Omega,\ k=NT,\ NT-1,\dots,1. \end{gathered}$$

The boundary conditions are

$$(3.9)\quad \begin{aligned} &\frac{\partial p}{\partial x}(a, y)=\frac{\partial p}{\partial x}(b, y)=0 && \text{for } y\in(c, d), \\ &\frac{\partial p}{\partial y}(x, c)-\alpha p(x, c)=0 && \text{for } x\in(a, b), \\ &\frac{\partial p}{\partial y}(x, d)=0 && \text{for } x\in(a, b). \end{aligned}$$

The equation (3.7) is again an ODE with respect to t for every $(x, y)\in\Omega$. Here $[x_{1,i}, x_{2,i}]=\Gamma_1\cap\overline{\Omega}_i$ for $i=1, 2,\dots, NS$. Moreover we mention that for our numerical tests we have considered $f=0$ and $l(v)=v$. In this case q becomes independent with respect to (x, y) and hence a vector is used to store the numerical values. A mathematical formula is available to integrate (3.7) :

$$(3.10)\quad q(t)=\int_t^T \eta(\tau)w(\tau)\exp(a_{22}(t-\tau))d\tau-\exp(a_{22}(t-T)),$$

$$\eta(\tau) = \sum_{i=1}^{NS} a_i \int_{x_{1,i}}^{x_{2,i}} p(x, c, \tau)dx. \tag{3.11}$$

Note also that the weights $\{a_i\}$ are stored in the vector $WEIGHT$.

The array P is initialized in the main program by

```
INITIALIZE-ARRAYS-2
Begin
    for i : = 1 to M+1 do
        for j : =1 to N+1 do
            p(i, j, NT+1) : =0
        next j
    next i
End ;
```

We give now the design of ADJST.

```
SUBROUTINE ADJST (u, p, qold, qnew, w, EPS, MAXITER, KODERR)
BEGIN
    for i : = 1 to M+1 do
        bdc (i)=0
    next i ;
    koderr : =0 ;
    iter : =1 ;
    while (koderr=0) and (iter≤MAXITER) do
        {compute p(qold) solving (3.6)}
        for k : =NT downto 1 do
            for i : =1 to M+1 do
                for j : =1 to N+1 do
                    rhs(i, j): =-(p(i, j,k+1)/DT+GDER(u(i, j, k))*
                                                      *qold(k))
                next j
            next i ;
            call ELLIPT (rhs, sol, bda, bdb, bdc, bdd) ;
            for i : = 1 to M+1 do
            for j : =1 to N+1 do
            p(i, j, k) : =sol (i, j)
            next j
            next i
        next k ;
        {compute qnew (p) according to (3.10)+(3.11)}
        {and || qnew-qold ||}
        for k : =1 to NT+1 do
            temp : =QUAD1(k,...) ;
            eta(k) : =temp*w(k)*exp(-a22*tknot(k))
        next k ;
        cst : =exp(-a22*T) ;
        err : =0 ;
        for k : =1 to NT do
            num : =NT+2-k ;
            for index : = 1 to num do
                itemp : = k+index-1 ;
                yaux(index) : = eta (itemp) ;
                taux (index) : = tknot(itemp)
                next index;
            ans : =QUAD2 (taux, yaux, num, taux(1), T) ;
            qnew (k) : = exp(a22*tknot(k))*(ans-cst) ;
            temp : = abs (qnew(k)-qold(k)) ;
            if err<temp
                then err : = temp
            end-if
        next k ;
        if err≤EPS
            then koderr : =1
            else iter : =iter+1
        end-if
    end-while
End {ADJST} ;
```


GDER — computes the derivative of function g ;

QUAD1 — integrates η given by formula (3.11) ; its parameters are dependent on the way the informations on subdomains Ω_i are given. We have used an array $IOMEGA$ (4, NS), where NS is the number of subdomains. If $\Omega_i=(x_{1i}, x_{2i}) \times (c, d)$ then we set
$IOMEGA$ (1, i) : = j_1, where j_1 is the index such that $xknot\ (j_1)= x_{1,i}$;
$IOMEGA$ (2, i) : = j_2, where $xknot\ (j_2)=x_{2,i}$;
$IOMEGA$ (3, i) : = 1 and $IOMEGA$ (4, i) : $=N+1$.

QUAD2 — computes the integral given in formula (3. 10). More precisely $QUAD2\,(t, y, n, a, b)$ returns an approximate value of $\int_a^b y(t)\,dt$ where vector t stores the knots $\{t_i\}$ and vector y stores the values $y_i=y(t_i)$, $i=1, 2,\dots,n$, for the numerical integration method.

The module HELPR0. Fitting the step ρ_j at each iteration is the most complicated point of the descent method. Such procedures are well known from the literature concerning minimization methods. However, they are considered for (finite dimensional) functions given by explicit formulas and no computational complexity considerations are made. In the case of control problems the situation is much more complicated since to evaluate the cost functional for a given control involves the computation of the corresponding state. Hence the state system has to be solved. The cost functional formula contains usually an integral and the straightfor ward conclusion is that its evaluation for a given control is time expensive. It follows that we need an algorithm which determines ρ_j using few such evaluations. Such an algorithm ([3], pp. 14—15) is presented here : consider $F(u_j-\rho_j d_j)=\min\{F(u_j-\rho d_j) ;\ \rho\geq 0\}$. We have to minimize the onedimensional function $s(\rho)=F(u_j-\rho\, d_j)$ for $\rho\geq 0$. We approximate s by a quadratic polynomial. Let δ be a starting value for the search process (in the case of ρ_j we consider an actualized value of ρ_{j-1}). Denote $s_0 = F(u_j)$, $s_1 = F(u_j - \delta d_j)$, $s_2 = F(u_j - 2\delta d_j)$ and S the polynomial which interpolates s with respect to 0, δ and 2δ. We have

$$S(\rho) = (2\delta^2)^{-1}(s_0 - 2s_1 + s_2)\rho^2 - (2\delta)^{-1}(3s_0 - 4s_1 + s_2)\rho + s_0.$$

If $s_0 - 2s_1 + s_2 > 0$ then

$$\rho^* = \frac{\delta}{2} \cdot \frac{3s_0 - 4s_1 + s_2}{s_0 - 2s_1 + s_2}$$

is the minimum point for S and we choose ρ_j such that

$$s(\rho_j) = \min\,(s(\delta),\ s(2\delta),\ s(\rho^*))$$

(it can be shown that $\rho^* > 0$).

If $s_0 - 2s_1 + s_2 < 0$ then ρ^* is the maximum point for S. Hence for $s_0 - 2s_1 + s_2 \leqslant 0$ we choose ρ_j such that

$$s(\rho_j) = \min\,(s(\delta),\ s(2\delta)).$$

Moreover, in our case the cost functional has the form $\Phi(w) = \varphi(w) + \psi(u_2)$ where

$$\varphi(w) = \int_0^T h(w(t))dt \quad \text{and} \quad \psi(u_2) = \iint_\Omega u_2(x, y, T)dxdy.$$

In order to evaluate $\Phi(w - \rho d)$ for a given ρ we have to solve the state system since we need also the corresponding value of ψ. To reduce the complexity of this procedure we take into account the fact that ψ is linear and therefore

$$\Phi(w^{(j)} - \rho d) = \varphi(w^{(j)} - \rho d) + \psi(v_2^{(j)}) - \rho\psi(v),$$

where $u_2^{(j)} - \rho v$ is the variation of $u_2^{(j)}$. The first two terms are immediately computed (via numerical integration methods) for every ρ. Let $u_1^{(j)} - \rho u$ be the variation of $u_1^{(j)}$. It follows that u is the solution of the system

$$(3.12)\qquad \begin{cases} \partial u/\partial t - \Delta u + a_{11}u = 0 & \text{in } Q, \\ u(x, 0) = 0 & \text{for } x \in \Omega, \\ \partial u/\partial \nu + \alpha u = d^{(j)} \sum\limits_{i=1}^{NS} a_i \int\limits_{\Omega_i} u_2^{(j)}\,dx + (w^{(j)} - \rho d^{(j)}) \sum\limits_{i=1}^{NS} a_i \int\limits_{\Omega_i} v dx & \text{in } \Sigma_1, \\ \partial u/\partial \nu = 0 & \text{in } \Sigma_2 \end{cases}$$

(recall that $d^{(j)} = \nabla\Phi(w^{(j)})$), while the system corresponding to v is approximated by

$$(3.13)\qquad \begin{cases} \partial v/\partial t + a_{22}v = g'(u_1^{(j)})u & \text{in } Q, \\ v(x, 0) = 0 & \text{for } x \in \Omega. \end{cases}$$

The system (3.12) is similar to (3.1) and (3.13) is similar to (3.2). Hence we compute v solving the coupled systems (3.12)+(3.13) by a "module" which

is quite similar to *STATE*. We call it *HELPR0* and we omit its design. Now $\psi(v)$ can be found. Hence we can compute $\Phi(w^{(j)} - \rho d^{(j)})$ for different values of ρ without calling *STATE* for each new value. Only *HELPR0* is called once.

Tridimensional arrays *UH*, *VH* and *VRO* are used by the routine *HELPRO*. They are initialized in the main program by *INITIALIZE–ARRAYS-3*

```
Begin
    for i := 1 to M+1 do
        for j := 1 to N+1 do
            UH(i, j, 1) := 0;
            VH(i, j, 1) := 0;
            for k := 1 to NT+1 do
                VRO(i, j, k) :=0
            next k
        next j
    next i
End;
```

The main program has the following design:

```
BEGIN
{Insert here blocks INITIALIZE–ARRAYS–1/2/3}
{Insert initialization for WOLD(*), WNEW(*), QOLD(*), QNEW(NT+1)}
R0 : == roint;
    iter := 1;
    KODEND := 0;
    while KODEND = 0 do
        call STATE (u, vold, vnew, wold, EPS1, MAXIT1, koderr)
        call ADJST (u, p, qold, qnew, wold, EPS2, MAXIT2, koderr)
        for k := 1 to NT do
            qold(k) := qnew (k)
        next k;
        call FIDER (vnew, p, wold, grad, TOL, koderr)
        {compute the l2-norm of the gradient vector grad}
        for k := 1 to NT+1 do
        ps(k) := grad(k) ↑ 2
        next k;
        ans := QUAD2 (tknot, ps, NT+1, tknot(1), tknot (NT+1));
        DL2 := SQRT(ans);
        {Convergence tests}
    if DL2<eps
    then KODEND := 1
    else iter := iter+1;
    if iter>MAXITER
    then KODEND := 9
    else {Compute current value of gradient step ρ}
            for i := 1 to M+1 do
                for j := 1 to N+1 do
```

```
                    TAB(i, j) : = vnew (i, j, NT+1)
               next j
     next i;
     PSIU2 : = VALMED(TAB, DX, M+1, N+1);
     call HELPRO (UH, VRO, VH, u, vnew, wold, grad, RO,
                    EPS3, MAXIT3, koderr);
     for i : = 1 to M+1 do
               for j = 1 to N+1 do
                    TAB(i, j) : = VH(i, j, NT+1)
               next j
     next i;
     PSIV : = VALMED(TAB, DX, M+1, N+1);
     roc : = RO;
     zo : = FI(0, wold, grad, PSIU2, PSIV);
     while (FI(roc,...)≥zo) and (KODEND=0) do
               roc : = roc/4;
               if roc*DL2<eps
               then KODEND : = 2
               end-if
     end-while;
     if KODEND = 0
     then z1 : = FI(roc,...);
          z2 : = FI(2*roc,....);
     temp : = z0+z2-2*z1;
     if z1<z2
       then RO : = roc;
            cost : = z1
       else RO : = 2*roc;
            cost : = z2
     end-if
     if temp>0
       then rotemp : =0.5*roc*(3*z0+z2-4*z1)/temp;
            ztemp : = FI(rotemp,...);
            if ztemp<cost
                 then RO : = rotemp;
                      cost : = ztemp
            end-if
     end-if
          {Compute wnew and the approximations}
          {(UH, VH) for (u, vold)}
          for k : = 1 to NT+1 do
               wnew(k) : = wold (k)-RO*grad (k);
               wold (k) : = wnew (k);
               for i : = 1 to M+1 do
               for j : = 1 to N+1 do
               u(i, j, k) : = u(i, j, k)-RO*UH(i, j, k);
               temp : = vnew (i, j, k)-RO*VH(i, j, k);
               vnew (i, j, k) : = temp;
```



```
                    vold (i, j, k) : = temp;
                    VRO (i, j, k) : = VH(i, j, k)
                 next j
               next i
             next k
           end-if
         end-if
       end-if
     end-while
   END.
```

The meaning of the variable *KODEND* is the following one :

= 0 — the algorithm is still working (no answer);

= 1 — convergence obtained in the sesnse that the l_2-norm of the gradient is "small";

= 2 — convergence obtained in the sense that the l_2-norm of the difference between two consecutive controls is "small";

= 9 — the algorithm is not convergent, i.e., no convergence conditions are satisfied after *MAXITER* iterations.

Note that the program can stop by nonconvergence or errors in a subroutine (see variables koderr). This possibility is not treated in the above description of the program in order to keep it simpler.

— The function $FI(\rho,...)$ computes the value of the cost functional corresponding to the control given by $wold(*)-\rho*grad(*)$.

— The subroutine *FIDER* returns the gradient values given by

$$\nabla\Phi(w^{(j)})\,(t) = -2/(w^{(j)}(t))^3 - \sum_{i=1}^{NS} a_i \iint_{\Omega_i} u_2^{(j)}(x, y, t)dxdy \cdot \int_{x_{1,i}}^{x_{2,i}} p_1^{(j)}(t, x, 0)dx. \tag{3.14}$$

Of course $grad(k)=\nabla\Phi(w^j)\,(t_k)$. The integral over Ω_i is computed using a Simpson type formula. We mention also that it is easy to demonstrate that the sequence $\{w^{(j)}\}$ does not converge to zero (see 3.14) since this would contradict the descent property of the gradient method.

— The function *VALMED* is computing the integral over Ω from $v(.,.,NT+1)$ using the values stored by the array *TAB*. A Simpson formula is used.

4. Numerical behavior of the algorithm. The numerical behavior of the algorithm is illustrated in the following table.

Descent method iteration	Number of iterations			$\Vert w^{(j+1)}-w^{(j)}\Vert_\infty$	$\vert\Phi(w^{(j+1)})-\Phi(w^{(j)})\vert$
	STATE	ADJST	HELPRO		
1	5	3	4	0.246	—
2	3	2	4	0.217E-1	0.26E-1
3	1	2	3	0.519E-2	0.14E-3
4	1	1	2	0.319E-3	0.20E-4

$xE\text{-}p$ means x times 10^{-p}

Note that the numerical results agrees with well known results on epidemics (a numerical simulation of the asymptotic behavior of the state system is also reported in [1]). They have been obtained in 6 sexadecimal digits/number arithmetic on IBM 4381 and IRIS 50.

Acknowledgement. The author is greatly indebted to dr. Octavian Petruş from the Computational Physics Laboratory of the University of Iaşi for his support in programming the numerical tests.

REFERENCES

1. Arnăutu, V., Barbu, V., Capasso, V. — *Controlling the Spread of a Class of Epidemics*, JAMO 20(1989), 297—317
2. Capasso, V. — *Asymptotic Stability for an Integrodifferential Reaction-diffusion System*, J. Math. Anal. Appl. 103 (1984), 575-588.
3. Legras, J. — *Algorithmes et programmes d'optimisation non lineaire avec contraintes. Application au controle optimal*, Masson, Paris, New York, Barcelone, Milan, 1980.

Received 1.VI.1989

Department of Mathematics
University „Al. I. Cuza" of Iaşi,
6600 — Iaşi, România

An. şt. Univ. „Al. I. Cuza" Iaşi
Informatică, 35 (1989)

SYSTOLIC ARRAYS FOR NUMERICAL INTEGRATION WITH ROMBERG'S FORMULA

BY

OCTAV BRUDARU

1. Introduction. Numerical integration appears frequently as a subproblem in many scientific computations and developing parallel architectures for such calculation is of real importance.

Some systolic arrays ([7]) implementing various numerical integration methods are given in [1—3].

In this paper, we present systolic implementations of Romberg's formula. We outline the method in Section 2 and discuss the obtaining of the starting values in Section 3. In Section 4 we present a triangular systolic array, while Section 5 deals whith the design of two linear systolic arrays. The extracting of extrapolated values is discussed in the last section.

2. Outline of the method. Let us suppose that we have to estimate $\int_a^b f(x)dx$. Romberg's method of approximate integration ([6, p. 107], [9, p. 114]) involves the following stages.

Stage 1. Compute the starting values

$$T_0^k=[f(a)+f(b)+2\sum_{j=1}^{2^k-1} f(a+jh_k)]h_k/2, \tag{2.1}$$

with $h_k=(b-a)/2^k$, $k=0, 1,\ldots, M$, where if $f\in C^m[a, b]$ then $2(M+1)<m$ ([6, p. 107]).

Stage 2. Solve the recursion

$$T_{i+1}^k=(4^{i+1}T_i^{k+1}-T_i^k)/(4^{i+1}-1), \quad k=0,\ldots,M-i-1, \quad i=0,\ldots,M-1. \tag{2.2}$$

If $e>0$ is the required tolerance then the computation is stopped if

$$|T_{i+1}^{M-i-1}-T_i^{M-1}|\leqslant e \tag{2.3}$$

for some $i\in\{1, 2,\ldots, M-1\}$.
In the opposite case, the process is continued for the next value of M.

3. Obtaining of starting values. Further, the clok tick (CT) is the time to performe a division or both a multiplication and an addition. We designate by t the time wich is a count of the number of CT s. We supose that each processing element (PE) of the network is active during each pulse number. Also, $L(t)$ designates the value circulating through the port L during t-th CT ($L(t)=$