Optimization of a Reparable Multi-State System (Part 2)

Adam Vaccaro

8228-44-9844

Abstract:

Herein, analytical and algorithmic solutions are provided for the optimal control input of a reparable dual-state system through time. The question is reformulated as a linear quadratic programming problem and a weighted sum approach is used to find an analytical solution of **u**\* while an algorithmic counterpart is found with the Lagrangian algorithm. The results were found to be J(**u**\*) = 0.004128820337337, p0,20 = 0.805579096183229, and p1,20 = 0.194420903816771. A sensitivity analysis was performed on the algorithm and it was found that while the magnitude of **u**(0)

1. Problem

In this problem, we are asked to find the optimal maintenance routine **u**\* for a reparable multi-state system characterized by two modes of operation: functional (mode 0) or failure (mode 1). Chung (1981) provided the mathematical model of a reparable multi-state device with multiple modes of failure, and from his initial investigation, we procure a pair of ordinary differential equations to represent our simplified two-state model:  
 dp0(t)/dt = -λ0p0(t) + μ1p1(t)

dp1(t)/dt = - μ1 p1(t) + λ0p0(t) t E [0,T0] (1)

with initial conditions

p0(0) = 1 p1(0) = 0 (2)

where p0(t) is the probability the system is in functional mode at time t, p1(t) is the probability that the system is in failure mode at time t, λ0 represents a constant failure rate for the device when it is in functional mode 0, and μ1 is the constant repair rate when the device is in failure mode 1.

To this system we add a control input function u(t), which represents the maintenance routine chosen to guide the system to a desired probability distribution at the end time T0:

dp0(t)/dt = -λ0p0(t) + μ1p1(t)+u(t)

dp1(t)/dt = - μ1 p1(t) + λ0p0(t)-u(t) (3)

The question of finding the optimal control input u(t) that will minimize the both the state signal and the control signal while steering the system to a desired probability distribution at T0 can be restated as a constrained optimization problem:

Minimize   
J(p0(t);p1(t);u(t)) = ||p0(t)-p0\*||22 + ||p1(t)-p1\*||22 + ||u(t)||22   
 (4)

over the time interval [0, T0] with initial conditions (2) and subject to constraints defined by system (3). Here, the objective function, J, is the sum of the state and control signals, p0\* is the desired probability of the system being in functional mode 0, and p1\* is the desired probability of the system being in failure mode 1. We set T0 = 1, λ0  = .3,and μ1 = .65. We assume the desired probability distribution is p0\* = 85% and p1\* = 15%. The problem is now to solve for an optimal maintenance routine u(t) that will minimize the objective function (4).  
  
The controlled system (3) can be reduced using the backward Euler’s method for solving ordinary differential equations and the system can be restated as a set of discretized algebraic equations:

(p0(ti+1) – p0(ti))/ Δt = -0.3p0(ti) + 0.65p1(ti) + u(ti)

(p1(ti+1) – p1(ti))/ Δt = -0.65p1(ti) + 0.3p0(ti) - u(ti) (5)

We now divide the time interval [0,T0] = [0,5] into 100 equal subintervals [ti=1,ti], i = 1,2,…,100. Set Δt = 1/20 and ti= iΔt. Thus, 0 = t0 < t1 < … < ti< … < t100 = 5. Let p0,i = p0(ti), p1,i=p1(ti), ui=u(ti), and **u** = [u0, u1, … , u100]T. The system of algebraic equations (5) can now be rewritten as:

(p0,i+1-p0,i)/(.05) = -0.3p­0,i+1+0.65p1,i+1+ui+1

(p1,i+1-p1,i)/(.05)= -0.65p­0,i+1+0.3p1,i+1-ui+1 (6)

If we let P­0 = [p0,1, p0,2,…,p0,100]T, P1 = [p1,1, p1,2,…,p1,100]T, and **u** = [u1, u2,…, u100] thenthe objective function can be restated:

J(P0;P1;**u**) = ||P0 – P0\*||22 + ||P1 – P1\*||22 + ||**u**||22 (7)

where P0\* = [85%, 85%,…, 85%]T100x1 and P1\*=[15%,15%,…,15%]T100x1.

From here, I begin to solve for the optimal control input **u\***.

**2. Methodology**

2.1 Analytical solution

To solve for the optimal control sequence **u\***, I used the weighted sum approach to the linear quadratic regulator (LQR) problem as presented by Chong and Zak (Chong 2013). First, I rewrote the objective function (7) and its associated constraints (6) to match the form of Chong’s LQR example (Chapter 20, example 20.10). To this end, I first introduced a change of variables:

xi = p0,i -.85

yi = p1,i -.15 (8)

Here it should be noted that:

xi = p0,i-.85

= (1-p1,i) -.85

= (1-(yi+.15)-.85

= -yi

in other words:

xi = -yi (9)  
Using these new variables, x and y, the objective function J (7) can be rewritten as:

J(X;Y;**u**) = ||X||22 + ||Y||22 + ||**u**||22 (10)

where X = [x1, x2,…, x100]T and Y = [y1, y2,…, y100]T.

Equation (10) can be expanded using the definition of the Euclidean norm:

J(X;Y;**u**) = ||X||22 + ||Y||22 + ||**u**||22

= (√Σxi2)2 + (√Σyi2)2 + (√Σui2)2

= Σxi2 + Σyi2 + Σui2

Now, we can use (9) to rewrite this as:

= Σxi2 + Σ(-xi)2 + Σui2

= 2(Σxi2) + Σui2

= (**½**)Σ(4xi2 + 2ui2)

Thus, the objective function J has been reformulated as:

J(X,**u**) = (**½)**Σ(4xi2 + 2ui2) (11)

Next, we use (8) to rewrite the constraints (6) as functions of x and y:

(xi+1-xi)/(0.5) = -0.3(xi+1+.85)+0.65(yi+1+.15)+ui+1

(yi+1-yi)/(0.5) = -0.65(xi+1+.85)+0.3(yi+1+.15)-ui+1

After expanding out the above equations and using relationship (9), the system of constraints can be reduced to a single equation:

xi+1 = .97561xi + .04878ui+1 (12)

where x0 = (p0,0 -.85) = 1 - .85 = .15

At this point, the optimization problem can be rewritten as a quadratic programming problem as demonstrated in Example 20.10 (Chong 2013). We now consider a similar linear programming problem:

minimize J = (**½**)**z**T**Qz** (13)

subject to **Az** = **b**

where **Q**, **A**, **z**, and **b** are as defined in Example 20.10, except with q = 4, r = 2, a = .97561, b = .04878, z0 = .15, and N = 100.

Chong provides the solution:

**z\*** = **Q**-1**A**T(**AQ**-1**A**T)-1**b**.

The second one hundred entries of **z\*** represent **u\***, the optimal control input;

P0 is obtained by adding .85 to the first one hundred entries of **z\***; and P1 can be found using the identity p1,I = 1-p0,I.

2.2 Algorithmic solution

An algorithmic solution to the problem was found using the Lagrangian Algorithm introduced by Chong (Ch. 23, 2013). I began by rewriting the

**3. Results**

3.1 Analytical results

The analytical solution yielded a value of J = 0.4335. The probabilities, P0 and P1, and the optimal control input **u\*** found with this analytical solution are plotted below in Figure 1.

Fig. 1: (top) The probabilities P0 (red) and P1 (blue) found using the analytical solution. (bottom) Analytical solution of **u\***.

3.2 Algorithmic results

**4. Observations and Conclusions**

The system is conservative:

The sum of p0 and p1 equals 1 at each time step. This property is expected of probability distributions and this result helps show that the algorithm is running properly.

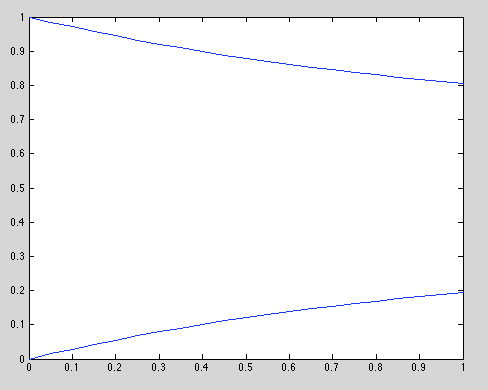


Fig. 1. p0 (top) and p1 (bottom) plotted over the interval [0,1]

Rate of convergence:

The DFP algorithm converged to the minimizer **u**\* in one step. On the second step, the algorithm yielded a solution vector **u**(2) that was very minimally different from **u**(1), but still lead to an identical value of J(**u**). For example, when the algorithm is run with initial parameters **u**(0) = **0** and H(0) = I20x20:

**u**(2) – **u**(1) = 1.0e-18 \* [0.2168, 0, 0.2168, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.4337, 0, 0, 0]T

and yet J(**u**(2)) - J(**u**(1)) = 0; i.e., J(**u**(1)) = J(**u**(2)).

Chong and Zak (2013) mention that a weakness of the DFP algorithm when applied to larger non-quadratic problems is that the algorithm has the tendency to get “stuck”. The phenomenon is attributed to Hk becoming nearly singular. In our case, the matrix H1 becomes nearly singular, and yields **u**(2) which ostensibly minimizes the objective function just as well as **u**(1). H2 becomes singular, and the algorithm ceases to work. So perhaps the phenomenon of J(**u**(1)) and J(**u**(2)) yielding the same value is an example of the algorithm getting “stuck”, only in this case it happens to be getting stuck on the proper minimizer. Furthermore, the fact H2 becomes singular implies that running the algorithm for more than two steps is unnecessary.

Sensitivity to H(0):

The algorithm was run with initial condition **u**(0) = **0** with positive definite symmetric H(0)’s of varying sizes. The magnitude of H(0) did not affect the values of J(**u**\*), p0,20, or p1,20.

|  |  |  |  |
| --- | --- | --- | --- |
| ||H(0)|| | J(**u**\*) | p0,20 | p1,20 |
| 1 | 0.004128820337337 | 0.805579096183229 | 0.194420903816771 |
| 1/1000 | 0.004128820337337 | 0.805579096183229 | 0.194420903816771 |
| 1/10 | 0.004128820337337 | 0.805579096183229 | 0.194420903816771 |
| 10 | 0.004128820337337 | 0.805579096183229 | 0.194420903816771 |
| 1000 | 0.004128820337337 | 0.805579096183229 | 0.194420903816771 |

Table 1. Shows the values of J(**u**\*), p0,20, and p1,20 when H(0)’s of different sizes are used. As can be seen, the magnitude of H(0) has no effect on the algorithm’s effectiveness.

The algorithm was also run with initial condition **u**(0) = **0** and indefinite symmetric H(0)’s of varying sizes. It turns out H(1) is always singular and the algorithm is only able to produce one iteration **u**(1) such that J(**u**(1)) = 0.004141982319579, a number relatively close to the minimizer. Here again, the magnitude of H(0) had no effect on the algorithm’s results.

The only requirement for H(0) in order to ensure the algorithm works properly is that it must be a symmetric positive definite; any symmetric positive definite matrix H(0) will suffice.

Sensitivity to **u**(0):

The algorithm was run with initial condition H(0) = I20x20 and **u**(0)’s of varying sizes. The magnitude of **u**(0) was shown to directly affect the end values of J(**u**\*), p0,20, or p1,20. As it turns out, **u**(0) taken as the initial guess yields the smallest J(**u**\*). As the size of **u**(0) increases, the algorithm loses its accuracy. When ||**u**(0)|| is sufficiently large, the algorithm produces nonsensical solution, for example, when ||**u**(0)|| = 4472.135, p0,20 = -1.11 and p1,20 = 2.11, an invalid probability distribution.

|  |  |  |  |
| --- | --- | --- | --- |
| ||**u**(0)|| | J(**u**\*) | p0,20 | p1,20 |
| 0 | 0.004128820337337 | 0.805579096183229 | 0.194420903816771 |
| 0.000447 | 0.004128820365409 | 0.805579102933662 | 0.194420897066338 |
| 0.447213 | 0.004155005780923 | 0.805381828059030 | 0.194618171940969 |
| 44.72135 | 0.267064606182265 | 0.786385729308510 | 0.213614270691491 |
| 4472.135 | 2629.465495215592 | -1.11323748938842 | 2.113237489388427 |

Table 2. shows the values of J(**u**\*), p0,20, and p1,20 when **u**(0)’s of different sizes are used. As can be seen, the magnitude of **u**(0) does indeed affect the algorithm’s ability to converge to the optimal solution.

5. References

E.K.P. Chong and S.H. Zak. An Introduction to Optimization: Fourth Edition.  
 Hoboken, New Jersey: John Wiley & Sons, 2013.

W.K. Chung. A Reparable Multi-state Device with Arbitrarily Distributed Repair   
 Times. *Micro. Reliabl.,* volume 21, number 2, pages 255-256, 1981.

6. Appendix

% math467DFP.m

% DFP ALGORITHM

%this script calls on functions symnorm.m, jvalue.m, and djvalue.m

addpath(genpath('/Users/adam/Dropbox/School/Fall 2013/MATH467'))

%% Inputs (could rewrite as function if desired)

u\_in = (1000)\*ones(20,1);

H\_in = eye(20,20);

nsteps = 3;

%% Initialize

%t = [0:1:20]; %time-steps

dt = 1/20; %time-step size

syms u0 u1 u2 u3 u4 u5 u6 u7 u8 u9 u10 u11 u12 u13 u14 u15 u16 u17 u18 u19;

U = [u0,u1,u2,u3,u4,u5,u6,u7,u8,u9,u10,u11,u12,u13,u14,u15,u16,u17,u18,u19];

%% solving for p0,20 and p1,20

p0(1)= dt\*(-.3\*1+.65\*0+U(1))+1; %first iteration/ p0,1

p1(1)= dt\*(-.65\*0+.3\*1-U(1))+0; %p1,1

for i = 2:20

p0(i) = dt\*(-.3\*p0(i-1)+.65\*p1(i-1)+U(i))+p0(i-1); %p0,2 ... p0,20

p1(i) = dt\*(-.65\*p1(i-1)+.3\*p0(i-1)-U(i))+p1(i-1); %p1,2 ... p1,20

end

p0\_f = p0(20); %p0,20

p1\_f = p1(20); %p1,20

%% rewrite J as function of U

K = (abs(p0(20)-.85)^2+abs(p1(20)-.15)^2+symnorm(U)^2); %K is the doubles version of J (not actually used in script)

%get rid of abs for sym functionality

J = sqrt((p0(20)-.85)^2)^2 +sqrt((p1(20)-.15)^2)^2+symnorm(U)^2;

%see jvalue.m to evaluate J of U

%see djvalue.m to evaluate dJ of U

%% rewrite J as quadratic function (solve for Q)

Q = jacobian(jacobian(J,U),U);

if isequal(Q,Q.') == 0

error('Q not symmetric')

end

%% DFP Algorithm

%Initial parameters

S(1).U = u\_in;

S(1).H = H\_in;

%algorithm loop

for k = 1:nsteps

S(k).g = djvalue(S(k).U);

if isequal(S(k).g, zeros(length(S(k).g))) == 1

error('g=0 at step %d', k)

else

S(k).d = -1\*S(k).H \*S(k).g; %search direction, d0

end

%solve for alpha:

S(k).alpha = (-S(k).g.'\*S(k).d)/(S(k).d.'\*Q\*S(k).d);

S(k).alpha = eval(S(k).alpha);

%next U

S(k+1).U = S(k).U + S(k).alpha\*S(k).d;

%del U

S(k).delU = S(k).alpha\*S(k).d;

%next g

S(k+1).g = djvalue(S(k+1).U);

%del g

S(k).delg = S(k+1).g - S(k).g;

%next H

S(k+1).H = S(k).H + S(k).delU\*S(k).delU.'/(S(k).delU\*S(k).delg.') - ((S(k).H\*S(k).delg)\*(S(k).H\*S(k).delg).')/(S(k).delg.'\*S(k).H\*S(k).delg);

end

%symnorm.m

%mimicks the norm function for symbolic objects

function n = symnorm(U)

ne = length(U);

for i = 1:ne

empty(i) = U(i)^2;

end

n = sqrt(sum(empty));

%jvalue.m

function [jval, p0\_fin, p1\_fin,p0,p1] = jvalue(U)

%computes the numeric value of the objective function J

%% solving for p0,20 and p1,20

dt = 1/20;

p0(1)= dt\*(-.3\*1+.65\*0+U(1))+1; %first iteration/ p0,1

p1(1)= dt\*(-.65\*0+.3\*1-U(1))+0; %p1,1

for i = 2:20

p0(i) = dt\*(-.3\*p0(i-1)+.65\*p1(i-1)+U(i))+p0(i-1); %p0,2 ... p0,20

p1(i) = dt\*(-.65\*p1(i-1)+.3\*p0(i-1)-U(i))+p1(i-1); %p1,2 ... p1,20

end

p0\_fin = p0(20);

p1\_fin = p1(20);

%% rewrite J as function of U

jval = (abs(p0\_fin-.85)^2+abs(p1\_fin-.15)^2+norm(U)^2);

%djvalue.m

%computes the numeric value for the gradient of the objective function J

function djval = djvalue(A)

syms u0 u1 u2 u3 u4 u5 u6 u7 u8 u9 u10 u11 u12 u13 u14 u15 u16 u17 u18 u19;

U = [u0,u1,u2,u3,u4,u5,u6,u7,u8,u9,u10,u11,u12,u13,u14,u15,u16,u17,u18,u19];

dt = 1/20;

p0(1)= dt\*(-.3\*1+.65\*0+U(1))+1; %first iteration/ p0,1

p1(1)= dt\*(-.65\*0+.3\*1-U(1))+0; %p1,1

for i = 2:20

p0(i) = dt\*(-.3\*p0(i-1)+.65\*p1(i-1)+U(i))+p0(i-1); %p0,2 ... p0,20

p1(i) = dt\*(-.65\*p1(i-1)+.3\*p0(i-1)-U(i))+p1(i-1); %p1,2 ... p1,20

end

J = sqrt((p0(20)-.85)^2)^2 +sqrt((p1(20)-.15)^2)^2+symnorm(U)^2;

djval = gradient(J,U);

djval = subs(djval, U, A);