Implementation of boundary conditions

Jérôme Hoepffner
hoepffner@irphe.univ-mrs.fr

June 2007

When discretizing partial differential equations, one has to implement boundary conditions. I present here a simple and general way to implement boundary condition. This method is useful when doing a matrix approach to the discretization, for instance in stability analysis: computing eigenmodes of a system, but as well for time marching, optimisation ...

In general, one can see boundary conditions as linear constraints on the state of our system. Thus, for n *independant* boundary conditions and a state variable of N degrees of freedom, one can remove n degrees of freedom, since they are *slaved* to the other degrees of freedom.

I will first present the general idea, valid for any boundary conditions, or in general for any state constraint, showing how we can decompose the state in degrees of freedom that we keep and degrees of freedom that we remove, how we can express the removed state variables as a function of the kept ones, thus preserving their action in the dynamical equations.

This method is as long to explain as simple to implement, so don't hesitate to often compare the description in §1 with the Matlab examples in §3.

1 Presentation

We express all boundary conditions and constraints in a matrix that we call C, we thus have

$$Cx = b,$$

where x is the vector of state variables, and b is a vector accounting for non-homogeneous boundary conditions. C is a $n \times N$ matrix with on each row a boundary condition, b is a $n \times 1$ column vector with on each row the value of the associated boundary condition. For instance considering a single homogeneous Dirichlet condition, C will be a zeros row vector, but with a 1 at the location of the boundary condition, for instance the first or the last position, and b will be zero, indicating that the state at this location should be zero. For a non-homogeneous boundary condition, b will be nonzero.

We can now decompose x into x_k, the part of x that we *keep*, and x_r the part of x that we *remove*. Reordering the state to emphasize this decomposition, we obtain

$$(C_k, C_r) \begin{pmatrix} x_k \\ x_r \end{pmatrix} = b, \tag{1}$$

where C has been reordered similarly to x. We will see in the matlab example, that this reordering is very simple to implement.

We can now express x_r as a function of x_k:

$$C_k x_k + C_r x_r = b \quad \Rightarrow \quad x_r = \underbrace{-C_r^{-1} C_k}_{G} x_k + \underbrace{C_r^{-1}}_{H} b. \tag{2}$$

In this expression, we have identified the matrix G, which we can call the "give-back" matrix, since it gives the removed degrees of freedom from the kept ones, and the H matrix, that accounts for the possible non-homogenous boundary conditions. or homogeneous boundary conditions, we would simply have $x_r = Gx_k$, thus G really being a *give-back* operator.

We have now done the first step, we have reduced the dimentionality of the state variable x, so that we have really $N - n$ independent degrees of freedom. We now use this expression to implement the boundary condition in our discretized partial differential equation, which we will consider for generality as a dynamic system (with a time derivative). If the time derivative was zero, like for instance when computing steady state solutions, nothing is changed, but the equations would be simpler. We have

$$E\dot{x} = Ax, \tag{3}$$

where $\dot{x}$ is the time derivative of the state variable, A is the dynamic operator (the discretized version of the PDE, also called as drift matrix in dynamical system, or stiffness matrix in finite element discretization), and E is the mass matrix (in finite element context). Note that E might be a singular matrix, in which case, (3) is known as a *descriptor system* or a diferential/algebraic system or as a singular system.

We know reorder E and A conformally to x to separate kept and removed variables to obtain

$$\begin{pmatrix} E_{kk} & E_{kr} \\ E_{rk} & E_{rr} \end{pmatrix} \begin{pmatrix} \dot{x}_k \\ \dot{x}_r \end{pmatrix} = \begin{pmatrix} A_{kk} & A_{kr} \\ A_{rk} & A_{rr} \end{pmatrix} \begin{pmatrix} x_k \\ x_r \end{pmatrix}.$$

We are not interested in describing the evolution of the removed degrees of freedom of the state, since we can at any time recover them from the kept degrees of freedom, so we hope to write an evolution equation for x_k only

$$E_{kk}\dot{x}_k + E_{kr}\dot{x}_r = A_{kk}x_k + A_{kr}x_r.$$

We now inject (2)

$$E_{kk}\dot{x}_k + E_{kr}(G\dot{x}_k + H\dot{b}) = A_{kk}x_k + A_{kr}(Gx_k + Hb),$$

that we sort as

$$(E_{kk} + E_{kr}G)\dot{x}_k + E_{rk}H\dot{b} = (A_{kk} + A_{kr}G)x_k + A_{kr}Hb),$$

and finally

$$\underbrace{(E_{kk} + E_{kr}G)}_{\tilde{E}}\dot{x}_k = \underbrace{(A_{kk} + A_{kr}G)}_{\tilde{A}}x_k + \underbrace{A_{kr}Hb - E_{rk}H\dot{b}}_{\tilde{F}},$$

where $\tilde{E}$ and $\tilde{A}$ are the mass matrix and stiffness matrix, modified to account for the boundary conditions, and $\tilde{F}$ is a (possibly time-varying) forcing vector accounting for non-homogeneous boundary conditions. Note that if the boundary conditions are not varying in time, then $\dot{b}$ will be zero, and that if the boundary conditions are homogeneous, then $\tilde{F}$ will be itself zero, in which case we simply have $\tilde{E}\dot{x}_k = \tilde{A}x_k$.

We can now do anything we like with this reduced version of the dynamic equations, like computing eigenvalues λ and eigenvectors v (with homogeneous boundary conditions)

$$\lambda \tilde{E} v = \tilde{A} v,$$

or computing a steady state x_s (with non-homogenous boundary conditions)

$$\tilde{E} \underbrace{\dot{x}_s}_{0} = \tilde{A}x_s + \tilde{F} \quad \Rightarrow \quad x_s = -\tilde{A}^{-1}\tilde{F}.$$

Once the kept degrees of freedom are obtained in any of these computations, we can recover the removed degrees of freedom using $x_r = Gx_k + Hb$.

2 Remarks

In most cases, we can apply the procedure described above without any problem, the only two rules being:

1. For Dirichlet or Neuman boundary condition, remove the mesh points at the location where the boundary condition applies.

2. For clamped boundary conditions (Dirichlet and Neuman at the same location), remove the mesh points *at* and *next to* where the boundary condition applies.

3 Example

In this section, I discuss a Matlab script that implement this method, on the three basic problems of eigenmodes, time marching and stationary state computation for the case of a beam with imposed boundary conditions. The dynamic equation is

$$\dot{x} = -\partial_{ssss} v.$$

The state variable x represents the position of the beam along the spatial direction $s \in [-1, 1]$. We have a fourth order derivative, we thus must impose four independent boundary conditions, in general

$$x(-1) = d_{-1}, \quad x(1) = d_1, \quad \partial_s x|_{-1} = n_{-1}, \quad \partial_s x|_1 = n_1.$$

We discretize x in space using chebyshev collocation (see appendix A), without *a priori* imposing any boundary conditions. We build the constraint matrix C,

```
N=60;
[D,s]=cheb(N-1);

I=eye(N);
c1=I([1,N],:); % dirichlet at first and last mesh point
c2=D([1,N],:); % Neuman at first and last mesh point
C=[c1;c2]; % the constraint matrix
```

we now define the degrees of freedom to be removed, for this we build the two vectors `k` and `r` that store the indices of mesh points to keep and to remove

```
r=[1,2,N-1,N]; % removed degrees of freedom
k=3:N-2; % kept degrees of freedom
```

We now build the *give-back* matrix G and H

```
G=-C(:,r)\C(:,k); % give-back matrix
H=inv(C(:,r)); % non-homogeneous contribution
```

and implement the decomposition in the dynamic equation

```
A=-D^4; % operator
AA=A(k,k)+A(k,r)*G; %implement boundary conditions
```

We are now set to compute the eigenmodes of the system. These modes tell about the oscillatory behaviour of the beam if for instance it is initially hit and then left alone to its intrinsic dynamics.

```
[Uk,S]=eig(AA); % compute eigenmodes
S=diag(S); [t,o]=sort(-real(S)); S=S(o); Uk=Uk(:,o); %sort
```

we recover the removed degrees of freedom and plot the 5 first eigenmodes:

```
Ur=G*Uk; % recover removed degrees of freedom
U=zeros(N,N-4); U(k,:)=Uk; U(r,:)=Ur;
plot(s,real(U(:,1:5)))
```

We now pass to the computation of the steady state for (time-invariant) non-homogeneous boundary condition, we chose for instance

$$x(-1) = -1, \quad x(1) = 1, \quad \partial_s x|_{-1} = -1, \quad \partial_s x|_1 = -1.$$

We build the boundary condition values vector b, and the corresponding forcing term $\tilde{F}$. Considering the steady state i.e. when x does not vary in time, we solve for x_s with zero time derivative, and plot the result

```
b=[1;-1;-1;-1]; % values of boundary conditions
FF=A(k,r)*H*b; % forcing term
xk=-AA\FF; % solve for steady state
xr=G*xk+H*b; % recover removed degrees of freedom
x=zeros(N,1);x(k)=xk;x(r)=xr;
plot(s,x)
```

A Spectral collocation

Here I put the Matlab code to compute the first order differentiation matrix using Chebyshev collocation. This comes from professor Nick Trefethen's book and home page (`http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/`).

```
function [D,x] = cheb(N)
% CHEB  compute D = differentiation matrix, x = Chebyshev grid
if N==0, D=0; x=1; return, end
x = cos(pi*(0:N)/N)';
c = [2; ones(N-1,1); 2].*(-1).^(0:N)';
X = repmat(x,1,N+1);
dX = X-X';
D  = (c*(1./c)')./(dX+(eye(N+1)));      % off-diagonal entries
D  = D - diag(sum(D'));                 % diagonal entries
```