Complementarity Problems · Week VI

CCO · Constraint Continous Optimization

Wednesday 7th October, 2009

Johan Sejr Brinch Nielsen

Email: zerrez@diku.dk
Cpr.: 260886-2547

Dept. of Computer Science,
University of Copenhagen

Contents

Introduction

This weeks assignment is to perform a physics simulation of balls falling and bouncing inside a box.

Linear Complimentary Problems

A linear complimentary problem (LCP) is a specific category of optimization problems, which take the form:

$$\begin{aligned} A\lambda + b &\geq 0 \\ \lambda &\geq 0 \\ \lambda^T(A\lambda + b) &= 0 \end{aligned}$$

Where A is a given matrix, b is a given vector and λ is the solution vector that is to be found.

Splitting Method

The Splitting method solves a LCP by first splitting the matrix A in two. Hereafter, it defines a series of sub-problems which also belongs to the LCP class. However, hopefully these are easy to solve.

The first step is to split A into to new matrices M and N such that:

$$A = M - N$$

The sub-problem is now defined as:

$$\begin{aligned} M\lambda^{k+1} + c^k &\geq 0 \\ \lambda^{k+1} &\geq 0 \\ (\lambda^{k+1})^T(M\lambda^{k+1} + c^k) &= 0 \end{aligned}$$

Where $c^k = b - N\lambda^k$. Letting $M\lambda^{k+1}$ be $M\lambda^k$ makes the left-hand side of the first inequality become:

$$M\lambda^k + c^k = M\lambda^k + b - N\lambda^k$$

$$= A\lambda^k + b$$

And the sub-problem is back to the original LCP. Hence, the only difference lies in using λ^{k+1} together with M and λ^k together with N. Since λ^k is the approximation of $\lambda^\star$ at iteration k, and λ^{k+1} that at iteration $k+1$, this is a fixed point computation yielding a solution to the original LPC.

Not surprisingly, this fixed point iteration can be solved iteratively. A bit more surprising is, that we need to apply the minimum map reformulation. This reformulation yields the following minimization problem:

$$\min_{\lambda}(\lambda^{k+1}, M\lambda^{k+1} + c^k) = 0$$

This formulation ensures the inequality constraints are fulfilled, since neither can be negative (the smallest of the two must be 0). This is equivalent to the equality constraint, that ensures the product of the two is 0. Hence, the new formulation is the same as the sub-problem.

We can now work on the new formulation:

$$\begin{array}{lll} \min & (\lambda^{k+1}, M\lambda^{k+1} + c^k) & = 0 \quad \Rightarrow \\ \min & (0, M\lambda^{k+1} + c^k - \lambda^{k+1}) & = -\lambda^{k+1} \quad \Rightarrow \\ \max & (0, -M\lambda^{k+1} - c^k + \lambda^{k+1}) & = \lambda^{k+1} \end{array}$$

The result is still a fixed point problem (it contains λ^{k+1} on both sides). However, there is one more trick up the sleeve: case analysis. I perform a case analysis on the sign of $S = -M\lambda^{k+1} - c^k + \lambda^{k+1}$:

Case 1: $S = -M\lambda^{k+1} - c^k + \lambda^{k+1} < 0$
If this is the case $\lambda^{k+1} = 0$, since $0 > S(x)$.

Case 2: $S = -M\lambda^{k+1} - c^k + \lambda^{k+1} \geq 0$
Clearly $S = \lambda^{k+1}$, since $S > 0$. That is:

$$-M\lambda^{k+1} - c^k + \lambda^{k+1} = \lambda^{k+1} \Rightarrow$$
$$-M\lambda^{k+1} - c^k = 0 \Rightarrow$$
$$-M\lambda^{k+1} = c^k$$

This situation is interesting. λ^{k+1} is now present on the left hand side only. Assuming that M is reversible, we can compute λ^{k+1} as:

$$\lambda^{k+1} = -M^{-1}c^k = -M^{-1}(b - N\lambda^k)$$

Combining the two cases yields the following closed term for λ^{k+1}:

$$\lambda^{k+1} = \max(0, M^{-1}N\lambda^k - b)$$

In order to solve the LCP, one can approximate on $\lambda^\star$ using above equation until an acceptable error has been reached.

Error Measure

The implementation uses the following residual to perform error measurement:

$$r = \min(A\lambda + b, \lambda)$$

This residual is derived from the equality constraint:

$$\lambda^T(A\lambda + b) = 0$$

Stating that either λ or $A\lambda + b$ must be zero when the solution in found. In all other cases, the smallest of the two is positive. Hence, it makes sence to define the error measure as the distance from this residual to zero. This distance is computed as:

$$e = r^T r$$

And the resulting error e is then used as a stop criteria in the iterative method.

Relation to Quadratic Problems

Any LCP can be reformulated as a quadratic problem (QP). The QP corresponding to the LCP in the previous section becomes:

$$\begin{aligned} \min_{\lambda} \quad & \lambda^T(A\lambda + b) & \\ \text{such that:} \quad & A\lambda + b & \geq 0 \\ & \lambda & \geq 0 \end{aligned}$$

Since $z \geq 0$ and $Mz + q \geq 0$, the product $z^T(Mz + q)$ must be positive. Hence, the minimum must be 0. The inequality constraints are still maintained, so the optimal solution will be equal to that of the original LPC.

Algebraic Transformation from LCP to QP

In this section I will convert the LCP solved in this weeks assignment to an QP. Let's start with the original LCP:

$$\begin{aligned} A\lambda + b &\geq 0 \\ \lambda &\geq 0 \\ \lambda^T(A\lambda + b) &= 0 \end{aligned}$$

Letting $h(\lambda) = A\lambda + b$, $x^\star = \lambda$ and $y^\star = h(\lambda)$ we obtain the following reformulation of the conditions:

$$\begin{aligned} h(\lambda) &\geq 0 \\ \lambda &\geq 0 \\ \lambda^T h(\lambda) &= 0 \\ y^\star &\geq 0 \\ x^\star &\geq 0 \\ (x^\star)^T h(\lambda) &= 0 \end{aligned}$$

The new conditions (last three) being repetitions of the original conditions (first three). Since $\min(\lambda, y^\star = A\lambda + b) = 0$ (see Splitting Method, page ii) we can add the constraint:

$$(y^\star)^T\lambda \quad = 0$$

Furthermore, since $x = \lambda$ and $y^\star = h(x)$ the following constraint is valid:

$$h(\lambda) + \nabla h(\lambda)^T\lambda - \nabla h(\lambda)^T\lambda - h(\lambda) = 0 \Rightarrow$$

$$h(\lambda) + \nabla h(\lambda)^T\lambda - \nabla h(\lambda)^T x^\star - y^\star = 0$$

The full set of conditions is now:

$$\begin{aligned} h(\lambda) &\geq 0 \\ \lambda &\geq 0 \\ \lambda^T h(\lambda) &= 0 \\ y^\star &\geq 0 \\ x^\star &\geq 0 \\ (x^\star)^T h(\lambda) &= 0 \\ (y^\star)^T\lambda &= 0 \\ h(\lambda) + \nabla h(\lambda)^T\lambda - \nabla h(\lambda)^T x^\star - y^\star &= 0 \end{aligned}$$

Which are the first order optimality conditions for the QP

$$\begin{array}{rrl} \min_\lambda & \lambda^T h(\lambda) & \\ \text{such that:} & h(\lambda) & \geq 0 \\ & \lambda & \geq 0 \end{array}$$

This transformation to QP shows that it is possible to solve LCP problems using QP methods.

Convergence Rate

Verification

Movie 1 - Falling Balls

Conclusion

Source Code

Run

```
close all;
clear all;
config = setup_config( 150, 300, 300, 10 );


lambda = 1;

iters = 1000;

M = moviein(iters);
fig = figure(1);

%set(fig, 'Renderer', 'OpenGL');

for i=1:iters
    info   = collision_detection(config);
    [config lambda] = integrate(config, info, 0.015, lambda);

    if mod(i, 4) == 0
        figure(fig);
        clf;

        hold on;
        draw_config( config );
        draw_info( config, info );
        hold off;
        axis square;
    end
    %name = strcat('plots_num/s',sprintf('%04d',i),'.png');
    %print('-dpng', name);

    %M(:,i) = getframe;
end

%mpgwrite(M, jet, 'movie.mpg');
```

Integrate

```
function [ config lambda ] = integrate(config, info, dt, lambda)

X = config.X;
Y = config.Y;
Vx = config.Vx;
Vy = config.Vy;
Fx = config.Fx;
Fy = config.Fy;
w  = config.W;

% Filter proximity info to only contain contact information
idx = info.D <= 0.01;
O  = info.O(idx==1,:);
Nx = info.Nx(idx==1,:);
Ny = info.Ny(idx==1,:);
D  = info.D(idx==1,:);

% Setup contact force problem
C       = length(D);
B       = length(w);
J       = zeros( C, B*2 );
%lambda = zeros( C, 1 );
if( C>0)
  for c=1:C
      a = O(c,1);
      b = O(c,2);
      J(c,a) =  Nx(c);
      J(c,b) = -Nx(c);
      J(c,a+B) = Ny(c);
      J(c,b+B) = -Ny(c);
  end
  u = [Vx; Vy];
  W = diag( [w; w] );
  A = J*W*J';
  b = J*u + dt*J*W*[Fx; Fy];
  lambda = solve_lcp(A,b,lambda);
```

```
end

% Compute the resulting contact forces in body-space
Fc = J'*lambda;
Cx = -Fc(1:B);
Cy = -Fc(B+1:end);

% Velocity update
Vx = Vx - w.*Cx + dt* w.*Fx;
Vy = Vy - w.*Cy + dt* w.*Fy;

% Position update
X = X + dt*Vx;
Y = Y + dt*Vy;

% Store new values in config object
config.X = X;
config.Y = Y;
config.Vx = Vx;
config.Vy = Vy;

end
```

Solve LCP

```
function lambda = solve_lcp(A, b, lambda)
% SOLVE_LCP - Solve the LCP problem
% input:
%    A    - A square symmetric positive (semi) definite matrix.
%    b    - The ``right-hand-side'' vector.
%  lambda - Initial guess of the solution.
%
% output:
%
%  lambda -  The resulting solution computed by the function.
%
% Copyright 2009, Kenny Erleben, DIKU:


    % Pseudo Code:
    % While not converged
    %  lambda = max(0, -inv( L+D)*(U*lambda + b)
    %  residual = min(A*lambda+b,lambda)
    %  error = residual'*residual
    % End

    fprintf(1, '>> SOLVER STARTED\n');
    % Expand Previous Lambda Vector To Fit A
    if (size(lambda, 1) < size(A, 1))
        lambda = [ lambda; zeros(size(A, 1) - size(lambda,1), 1) ];
    elseif (size(lambda, 1) > size(A, 1))
        lambda = lambda(1:size(A,1));
    end

    % Compute M and N
    M = tril(A);
    N = -triu(A,1);
    assert (all(all(A == (M-N))));

    % Compute Matrix Inverse Using LU Decomposition (more precise)
    [L1 U] = lu(M);
    Mi = inv(L1) * inv(U);

    % Compute Initial Error
    residual = min(A*lambda + b, lambda);
    error = residual' * residual;
    fprintf(1, 'Error: %d\n', error);

    count = 1;
    while error > 0.00001 && count < 20000
        newLambda   = max(0, Mi*N*lambda - b);
        max(newLambda)

        lambda = newLambda;
        residual = min(A*lambda + b, lambda);
        newError = residual' * residual;
        if not (error > newError)
            error = newError;
            break
        end
        error = newError;
        fprintf(1, 'Error: %d -> %d\n', count, error);
        count    = count + 1;
    end

end
```