Laser Pulse Shaping problem

Practical Assignments Natural Computing, 2009

Rick van der Zwet
<hvdzwet@liacs.nl>

19 december 2009

1 Introduction

The report is focused on the so-called *laser pulse shaping problem*. Today's lasers are also used within the range of atoms or molecule research. Using small pulses it is able to align and alter the movement of the atoms.

The problem lies in the fact the atoms cannot be controlled by any type of laser pulse. There are many parameters which could all be set to 'shape' the laser pulse the way it can move the atoms.

To turn and tweak all the 'knobs' at the same time there will be *Particle Swarm Optimizer (PSO)* used to explore the search space. The *PSO* is basically a whole bunch of individual agents which all try to find an optimum into the search space. During this search they get input about other potential bests from the whole swarm (broadcast style) and the neighborhood (observation) and using this values they determine their new location.

2 Problem description

A laser pulse going through a crystal produces light at the octave of its frequency spectrum. The total energy of the radiated light is proportional to the integrated squared intensity of the primary pulse. Explicitly, the time-dependent profile of the laser field in our simulations is given by:

$$E(t) = \int_{-\infty}^{\infty} A(\omega) exp(i\phi(\omega)) exp(i\omega t) d\omega, \tag{1}$$

where $A(\omega)$ is a Gaussian window function describing the contribution of different frequencies to the pulse and $\phi(\omega)$, the phase function, equips these frequencies with different complex phases.

To determine the best solution a fitness function is needed, which could be found in the shape of equation 2

$$SHG = \int_0^T E^4(t)dt \longrightarrow maximization \tag{2}$$

Note that $0 < SHG < 1$

3 Statistics

4 Approach

The Wikipedia page 'Particle swarm optimization' [1] contained a reference implementation used to implement the algorithm. The nice part about the algorithm is its flexibility in tuning. As within the *PSO* there are many 'knobs' which could be tweaked as well, like likeliness of heading for the global optimum, neighborhood optimum and local optimum.

5 Implementation

The code is written in *Octave*[2] which is the open-source 'variant' of *MATLAB*©[3]. There are small minor differences between them, but all code is made compatible to to run on both systems. The code is to be found in Appendix 8.

As work is done remotely, the following commands are used:

```
matlab-bin -nojvm -nodesktop -nosplash -nodisplay < %%PROGRAM%%
```

```
octave -q %%PROGRAM%%
```

The flock is represented into a 3d block. A slice of that block contains a local swarm, a column in slice is a individual particle.

6 Results

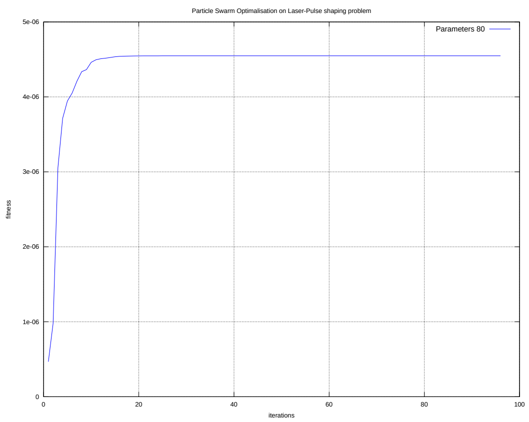
The program is run against a parameter set of 80. The algorithm is allowed to run for 10 minutes with a maximum of 1000 iterations. If there are no improvements after 5 iterations then it will bail out as well.

[1] http://en.wikipedia.org/wiki/Particle_swarm_optimization
[2] http://www.gnu.org/software/octave/
[3] http://www.mathworks.com/products/matlab/

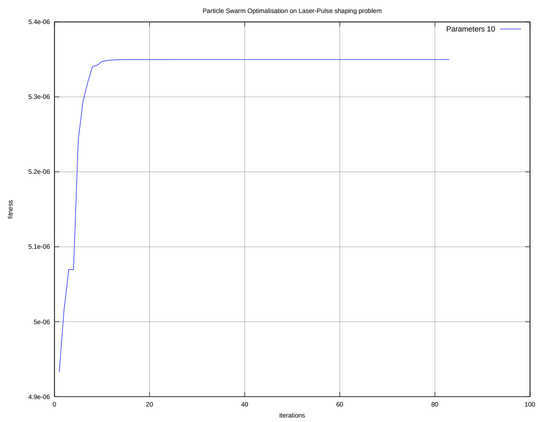
The algorithm is kind of 'social' e.g. it will favor the neighborhood and the global optimum more than it's own local optimum. Also its active, meaning it changes direction fast the moment the optimum is found somewhere else.

Size of the local swarms is 50, each with 10 agents.

After running 5 times, the best fitness found was 0.0000045481. Improvement is almost only shown in the first 100 iterations see figure 1a, afterwards it quickly stalls. Trying with a smaller number more and less show the same result as seen in figure 1b



(a) 80



(b) 10

Figuur 1: Fitness throughout the iterations

Changing various flags like walkspeed *wander* or changing the socialness of the agents does not prove to change much in the algoritm result.

7 Conclusions

Giving the lack of external results of other algorithms in large scale setups (80 parameters) its hard to say whether this is a good or worse preforming algorithm.

For furher research within the algorithm there are many knobs to tweak as well. One could think of implementing a algorithm around this setup as well.

8 Appendix 1

```
% Particle Swarm optimalisation
% BSDLicence
% Rick van der Zwet - 0433373 - <hvdzwet@liacs.nl>
% Modeled after http://en.wikipedia.org/wiki/Particle_swarm_optimization

% Dimention settings
parameters = 10;

% If global optimum does not change this many steps bail out
iteration_break = 5;
max_iterations = 1000;
max_time = 10 * 60; % in sec

% Flock properties
local_swarm_size = 50;
local_swarms = 10;

%% Particle properties
% Speed of walking around to a certain direction
wander = 0.4;

% 'Influence' of the envirionment with regards to solutions
% Trust the group global solution to be feasible
c_social  = 0.4;
% Trust the neighbor solution to be feasible
c_cognitive = 0.4;
% Trust the own best solution to be feasible
c_ego = 0.2;

% Variables used for plotting
fitness_history = [];
fitness_iterations = [];

% Initiate all particles
flock_p = rand(parameters,local_swarm_size,local_swarms) .* (2 * pi);
flock_v = zeros(size(flock_p));


% Global best placeholder
g_best = ones(parameters,1) .* 9;
```

```
g_fitness = 0;
% at (:,x) lives the neighbor best of local_swarm 'x'
n_best = ones(parameters,local_swarms) .* 9;
n_fitness = zeros(parameters,local_swarms);
% at (:,p,x) leves the local best of particle 'p' in local_swarm 'x'
l_best = ones(parameters,local_swarm_size,local_swarms) .* 9;
l_fitness = zeros(local_swarm_size, local_swarms);

idle_counter = 0;
tic();

% Code not optimised for performance, but for readablility
for i = 1:max_iterations
    for s = 1:local_swarms
        fitness = SHGa(flock_p(:,:,s));
        % See if we got any better local optimum
        for p = 1:local_swarm_size
            if fitness(p) > l_fitness(p,s)
                l_fitness(p,s) = fitness(p);
                l_best(:,p,s) = flock_p(:,p,s);
            end
        end

        % See if we got any better neighbor optimum
        for p = 1:local_swarm_size
            if l_fitness(p,s) > n_fitness(s)
                n_fitness(s) = l_fitness(p,s);
                n_best(:,s) = l_best(:,p,s);
            end
        end
    end

    idle_counter = idle_counter + 1;

    % See wether we have a new global optimum
    for s = 1:local_swarms
        if n_fitness(s) > g_fitness
          g_fitness = n_fitness(s);
          g_best = n_best(:,s);
          idle_counter = 0;
```

```
        end
    end

    % Stop conditions
    if idle_counter == iteration_break
        fprintf('Caught by idle_counter\n');
        break;
    end
    if toc > max_time
        fprintf('Caught by max_time used \n');
        break;
    end


    fprintf('%04i : %.15f\n', i, g_fitness);
    fitness_iterations = [fitness_iterations, i];
    fitness_history = [fitness_history, g_fitness];


    % Update particles to new value
    r_cognitive = rand();
    r_social = rand();
    r_ego = rand();
    for s = 1:local_swarms
        for p = 1:local_swarm_size
            flock_v(:,p,s) = flock_v(:,p,s) * wander + ...
              (g_best - flock_p(:,p,s)) * (c_cognitive * r_cognitive) + ...
              (n_best(:,s) - flock_p(:,p,s)) * (c_social * r_social) + ...
              (l_best(:,p,s) - flock_p(:,p,s)) * (c_ego * r_ego);
            flock_p(:,p,s) = flock_p(:,p,s) + flock_v(:,p,s);
        end
    end
end

% Dispay hack
g_fitness
g_best

plot(fitness_iterations,fitness_history);
```

```
title(sprintf('Particle Swarm Optimalisation on Laser-Pulse shaping problem'))
       ;
ylabel('fitness');
xlabel('iterations');
grid on;
legend(sprintf('Parameters %i',parameters));
print(sprintf('pso-fitness-%.10f.eps', max(fitness_history)),'-depsc2');
```

```
% Particle Swarm optimalisation
% BSDLicence
% Rick van der Zwet - 0433373 - <hvdzwet@liacs.nl>
% Modeled after http://en.wikipedia.org/wiki/Particle_swarm_optimization

% Iterate on multiple vectors
function fitness = SHGa(v)
    for i = 1:length(v(1,:))
        fitness(i) = SHG(v(:,i));
    end
end

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function calculates the Second Harmonic Generation
% of a Gaussian of frequencies with a given phase function.
%
%     phi - an input COLUMN vector, containing the phase function.
%     SHG - the output of the calculation; scalar.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [SHG] = SHG(phi);

%constant for consistency with the Fortran Calculation...
c_fortran = 153.7687;

% Generate the Gaussian and the phase function consistently.
Np = length(phi(:,1));
Nv = 4000;
v = linspace(-300,300,Nv); %Linearly Spaced Vector
G = 40;
Ain = exp(-(v/G).^2); %The Gaussian of Frequencies

%Distribute the phase function according to the desired resolution
step = round((2600-1400)/Np);
step = step + (step==1);
phase = zeros(1,Nv);
k = 1;
for j = 1400:step:2600-step+1,
    phase([j:j+step-1]) = phi(k);
    if (k < Np)
        k = k+1;
    else
        k = Np;
    end
end
    % *** The Core: SHG *** %

%Fourier Transform with phase shift (phi) on the Gaussian *Ain*
E_t = fftshift(ifft(fftshift(exp(i*phase).*Ain)));

%plot(abs(E_t));
%Integrate the result to yield the SHG
SHG = sum(abs(E_t).^4)/c_fortran;
```

```


%%%%%%%%%%%%%%%%%%%%%%%%%%% E O F %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```