All the following is subject to $\boldsymbol{w}, \boldsymbol{s} \geq 0$:

$$
\begin{aligned}
\mathcal{L} &= \min_{\boldsymbol{w},\boldsymbol{s}} ||M\boldsymbol{w}||^2 + \mu||\mathbf{1} - A\boldsymbol{w} - \boldsymbol{s}||^2 \\
&= \min_{\boldsymbol{w},\boldsymbol{s}} \boldsymbol{w}^T M^T M\boldsymbol{w} + \mu \left((\mathbf{1}^T - \boldsymbol{w}^T A^T - \boldsymbol{s}^T)(\mathbf{1} - A\boldsymbol{w} - \boldsymbol{s})\right) \\
&= \min_{\boldsymbol{w},\boldsymbol{s}} \boldsymbol{w}^T M^T M\boldsymbol{w} + \mu \left(\mathbf{1}^T\mathbf{1} - \mathbf{1}^T A\boldsymbol{w} - \mathbf{1}^T\boldsymbol{s} - \boldsymbol{w}^T A^T\mathbf{1} + \boldsymbol{w}^T A^T A\boldsymbol{w} + \boldsymbol{w}^T A^T\boldsymbol{s} - \boldsymbol{s}^T\mathbf{1} + \boldsymbol{s}^T A\boldsymbol{w} + \boldsymbol{s}^T\boldsymbol{s}\right) \\
&= \min_{\boldsymbol{w},\boldsymbol{s}} \boldsymbol{w}^T(M^T M + \mu A^T A)\boldsymbol{w} + \mu\boldsymbol{s}^T I^T I\boldsymbol{s} + \mu\left(1 - 2\mathbf{1}^T A\boldsymbol{w} - 2\mathbf{1}^T\boldsymbol{s} + 2\boldsymbol{w}^T A^T\boldsymbol{s}\right) \\
&= \min_{\boldsymbol{y}} \boldsymbol{y}^T \underbrace{\begin{pmatrix} M^T M + \mu A^T A & \mu A^T \\ \mu A & \mu I \end{pmatrix}}_{C^T C} \boldsymbol{y} - 2\begin{pmatrix} \mu\mathbf{1}^T A \\ \mu\mathbf{1}^T \end{pmatrix}\boldsymbol{y} + \underbrace{\mu\mathbf{1}^T\mathbf{1}}_{\boldsymbol{d}^T\boldsymbol{d}} \\
&= \min_{\boldsymbol{y}} \boldsymbol{y}^T H\boldsymbol{y} + \boldsymbol{f}\boldsymbol{y}
\end{aligned}
$$

where $\boldsymbol{y}$ is the $m+n$ vector $\begin{pmatrix} \boldsymbol{w} \\ \boldsymbol{s} \end{pmatrix} \geq 0$. Yet it still does not look like:

$$\min_{\boldsymbol{y}} ||C\boldsymbol{y} - \boldsymbol{d}||^2 = \min_{\boldsymbol{y}} \boldsymbol{y}^T C^T C\boldsymbol{y} - 2\boldsymbol{d}^T C\boldsymbol{y} + \boldsymbol{d}^T\boldsymbol{d}$$

because of M.

Listing 1: Solving minimization problem with the subset method

```
function [w, Aw, L] = compute_graph(X, kind, mu)
% All stories have a hero and our is not different.  So meet X, a handsome and
% brave set of n d-dimensional vectors.
[n, d] = size(X);
m = nchoosek(n, 2);
% She is in love with the equally beautiful U, a matrix containing all the
% m possible edges between X's nodes.
U = sparse(n, m);
% Yet for now, U, like many boys of his age, is quite empty.  Therefore X,
% must fill him with the weights in w.  But to be fair, she has no feasible
% solution to propose so far.
w = [];
% Fortunately, she will be helped by some friends, although they shall be
% presented later, as they are, with all due respect, mainly calculations'
% artifice (and as such, they don't have a clue about how to start).
M = sparse(d*n, m);
MAX_ITER = 50;
nb_iter = 0;
bin_upper = n*(0:n-1) - cumsum(0:n-1);
considered_last_time = [];

% One day, an old man told X that she could for instance start with this
% random small subset:  a third of (d+1)n edges, as this was indeed common
```

```matlab
% knowledge, provided by a book called Theorem 3.1, that U cannot contains
% more.  At this point, the astute reader may wonder why we do not use a more
% sensible initial choice like bind each node with its closest neighbor.  Well,
% I am telling the story so we do like this.  But feel free to contribute!
edges = randi(m, 1, floor(0.33*(d+1)*n));

% And thus begin the quest of X, until she can not add more edges to U or
% until she get fed up and realize that organizing illegal fights of turtles is
% much more exciting than finding love.
while (numel(edges) > 0 && nb_iter < MAX_ITER)
  % U was also quite stubborn and felt that linear indexing of edges
  % was not doing justice to his amazing 2D abilities.  Therefore X has
  % to resort to her cunning to convert them.  The edges 1 through n-1
  % were from i=1, those from n to n+(n-2)-1=2n-3 started at i=2
  % and so on.  It turns out that bin_upper has memorized all these
  % upper bounds so finding is was simply a matter of finding the
  % maximum possible bounds.
  vertex_i = arrayfun(@(x) find(x' <= bin_upper, 1, 'first'), edges) - 1;
  % Then j follows
  vertex_j = vertex_i + edges - bin_upper(vertex_i);
  % and the pairs (i,j) could be converted into U indexes
  positive = bsxfun (@(x,y) sub2ind(size(U), x, y), vertex_i, edges);
  negative = bsxfun (@(x,y) sub2ind(size(U), x, y), vertex_j, edges);
  % in order to represent the newly selected edges.
  U(positive) = 1;
  U(negative) = -1;
  A = abs(U);
  assert(sum(A(:))/2 <= (d+1)*n, 'there are too many edges');

  % To assess their compatibility, the tradition was to compute the
  % Frobenius norm of X times the graph Laplacian.  Both find this
  % method awkward and they choose to reformulate it as an Euclidean
  % distance.  But doing so require the help of a friend:  M.
  T = U'*X; % y^(k) = U^T x^k is thus the kth column of T
  % First X mixes her columns with U, producing d new vectors of
  % length n:  y^(k).
  for k=1:d
    first_row = 1 + (k-1)*n;
    last_row = n + (k-1)*n;
    % These new vectors were soon promoted as diagonal matrices and
    % filled M from top to bottom (although it would have been
    % faster to do it in parallel).
    Yk = spdiags(T(:,k), [0], m, m);
    M(first_row:last_row, :) = U*Yk;
  end

  % Having done all this preparatory work, X could finally go see an
  % oracle living in the mountain, the so called quadprog, and ask him to
  % set w optimally according to M.  (Actually, she had also heard of
```

```
    % another one, SDPT3, potentially faster and able to deal with sparse
    % matrix instead of converting M'*M to a full one and taking 2n^4
    % bytes of memory.  But she had to ask her question in a slightly
    % different language:
    % http://www.math.nus.edu.sg/ mattohkc/sdpt3/guide4-0-draft.pdf
    if strcmpi(kind, 'hard')
      % In one method, she had to ensured that the weighted sum of
      % degree was at least one for each node, or in the language of
      % the oracle:  -Aw <= -1.  But this was only possible
      % for the nodes that were part of at least one edge, that is
      % for the non zero rows of A).
      o = optimoptions(@quadprog, 'Algorithm', 'active-set', 'Display', 'final-detailed
          ');
      [w, f, flag, output, lambda] = quadprog(M'*M, sparse(m, 1), -A, -(sum(A, 2)>0),
          [], [], [], [], w, o);
      z = lambda.ineqlin;
      derivative = 2*M'*M*w - A'*z;
      save('out', 'M', 'w', 'A', 'lambda', 'derivative');
    else
      % There was another method were a portion alpha of the nodes
      % were allowed to have degree less than one.  But she still
      % has to think about to formulate
      % min_{w,s} ||Mw||^2 + mu||1 - Aw - s||^2
      % for quadprog or lsqnonneg (http://math.stackexchange.com/q/545280)
      error(strcat(kind, ' is not yet implemented'));
    end
    % Because the new (w, z) were supposed to be feasible solution,
    % dΛ/dw has to be positive.  Therefore, she finds the
    % edges where it was not the case to add them in the next step.
    [val, may_be_added]=sort(derivative(find(derivative<0)));
    % Of course maybe there was nothing to do.  Or more concerning, the
    % oracle was rambling and returned a solution that yields the same set
    % of edges to add as previously, in which case there was no point in
    % continuing any further.
    if ((length(may_be_added) == 0) || (length(may_be_added) == length(
        considered_last_time) && all(may_be_added' == considered_last_time)))
      break;
    end
    % She decide to add only half of them but probably there were other
    % ways of doing it (like adding the "smallest one" ?)
    edges = may_be_added(1:max(1, floor(end/2)))';
    considered_last_time = may_be_added;
    nb_iter = nb_iter + 1;
end
% When X finds the perfect weights for her graph (and hopefully not because
% she just give up), she have to fill some paperwork like computing weighted
% degree and Laplacian to make their union official.
Aw = A*w;
W = spdiags (w, [0], m, m);
```

```
L = U*W*U';
end
```

Listing 2: Computing hard graph

```
function [w, Aw, L] = compute_hard_graph(X)
  [w, Aw, L] = compute_graph(X, 'hard');
end
```

Listing 3: Computing α-soft graph

```
function [w, Aw, L] = compute_alpha_graph(X, alpha, tol)
[n, d] = size(X);
m = nchoosek(n, 2);
mu = 5*rand();
tau0 = 1.5;
MAX_ITER = 50;
% Set λ so that τ ≥ 1 with equality at MAX_ITER.
lambda = (tau0 - 1)/MAX_ITER;
can_improve = true;
while (can_improve)
  w, Aw, L = compute_graph(X, 'soft', mu);
  tmp = max(zeros(m, 1), ones(m, 1) - A*w);
  alpha_bar = tmp'*tmp/n;
  % Maybe we don't need this complication and keep a fixed τ = τ_0.
  tau = tau0/(1 + nb_iter*lambda);
  if (alpha_bar < alpha)
    % We want to increase ᾱ so we need decrease μ, which in
    % turn require τ ≤ 1
    tau = 1/tau;
  end
  % It is supposed to correspond to: "we then adjust μ up or down
  % proportionally to how far η(w)/n = ᾱ is from the
  % desired value of α."
  mu = tau*abs(alpha_bar - alpha)/alpha
  nb_iter = nb_iter + 1;
  can_improve = abs(alpha_bar - alpha) < tol && nb_iter < MAX_ITER;
end
end
```

Listing 4: Use the built graph to classify samples

```
function result = graph_classify(labelled, label, unlabelled)
n = numel(label);
u = size(unlabelled, 1);
X = [labelled; unlabelled];
[w, Aw, L] = compute_hard_graph(X);
beq = [label; zeros(u, 1)];
Aeq = [eye(n) zeros(n, u); zeros(u, n+u)];
% TODO look at the fast method suggested in the paper: Spielman, D. A.,
```

```
% Teng, S.-H. (2004).  Nearly-linear time algorithms for graph partitioning,
% graph sparsification, and solving linear systems.  Proc.  36th ACM STOC.
x = quadprog(L, [], [], [], Aeq, beq);
result = x(n+1:end) > 0.5;
end
```