Molecular Distance Geometry Problem and training of neural networks.

Exercize 1: Molecular Distance Geometry Problem

The *Molecular Distance Geometry Problem*, or MDGP, consists in reconstructing the 3-D structure of a molecule from information on the distances between its atoms. Let us assume that the distances between all pairs of atoms are known (with infinite precision). Let $D \in \mathcal{R}^{m \times m}$ the symmetrix matrix of the Euclidean distances d_{ij} for each pair $\{i, j\}$.[1]

Let $\underline{x}^1, \ldots, \underline{x}^m \in \mathbb{R}^3$ the unknown positions of the atoms. We can assume w.l.o.g. that the m-th point has coordinates $\underline{x}^m = (0, 0, 0)$. The problem consists of determining the coordinates $\underline{x}^1, \ldots, \underline{x}^{m-1} \in \mathbb{R}^3$ of the other $m - 1$ atoms so to satisfy the distances d_{ij} between the pairs of atoms i and j. The corresponding constraints are:

$$||\underline{x}^i - \underline{x}^j||_2 = d_{ij}, i, j = 0, 1, \ldots, m.$$

a) Give a nonlinear unconstrained optimization formulation for the problem.

b) Due to the non-convexity of the problem, implement a multistart method based on the methods implemented in the previous lab. Let:

- $\varepsilon > 0$ the tolerance of the multistart algorithm
- $\bar{f}$ the known optimal value of the objective function
- M the maximum number of multistart iterations

The multistart algorithm is as follows:

(a) Let $\underline{x}$ be a point in $\mathbb{R}^{3(m-1)}$. Let $\underline{x}^* \leftarrow \underline{x}$.

(b) If $f(\underline{x}^*) < \bar{f} + \varepsilon$ or if more than M iterations, the algorithm stops; otherwise, go to step c).

(c) Find local minimum $\underline{x}' = (x^1, \ldots, x^{m-1})$ from initial point $\underline{x}$, with a nonlinear optimization method (with a tolerance $\varepsilon' > 0$ and a maximum number of iterations M').

(d) If $f(\underline{x}') < f(\underline{x}^*)$ update $\underline{x}^* \leftarrow \underline{x}'$.

(e) Find new initial point $\underline{x} \in \mathbb{R}^{3(m-1)}$, randomly.

(f) Go to step b).

c) Apply the algorithm to the following instance with $m = 4$ atoms ($d_{ii} = 0$ for any i):

d_{ij}	2	3	4
1	1.526	2.491389536	3.837572036
2	0	1.526	2.491389535
3	-	0	1.526

[1] A reference is: J.M. Yoon, Y. Gad, Z. Wu, *Mathematical modeling of protein structure using distance geometry*, Technical report TR00-24, DCAM, Rice University, Houston, 2000, available from: `http://www.caam.rice.edu/caam/trs/tr00.html#TR00-24`.

To build the initial random solution, use `rnd.m`, that generates a vector of n elements between `-bound` and `bound` according to a uniform distribution.

```
function xrnd = rnd(n, bound)
  xrnd = zeros(n,1);
  for i = 1:size(xrnd,1)
    xrnd(i) = (rand()-0.5)*2*bound;
  end
end %of function
```

d) Apply the Gauss-Newton Method for the solution of nonlinear least square problems. We give `jac.m`, that computes the Jacobian at `x` of a vector function `r` with `m` components. The second parameter represents the number of components.

```
% Jacobian of a vector function at a point
function J = jac(r, m, x)
  n = length(x);
  J = zeros(m,n);
  h = 0.0001;

  for i = 1:m
   for j = 1:n
     delta = zeros(n, 1); delta(j) = h;
     rd=r(x+delta);
     rx=r(x);
     J(i,j) = (rd(i)  -  rx(i)) / h;
   end
  end
end %end of function
```

Use Matlab `pinv`, that, given matrix A, computes its pseudoinverse $\left(A^TA\right)^{-1}A^T$.

Gauss-Newton method It is a variant of Newton method for nonlinear least squares problem. Consider:

$$f(\underline{x}) = \sum_{i=1}^{m}(r_i(\underline{x}))^2, \tag{1}$$

where $\underline{x} \in R^n$ and $r(\underline{x}) = (r_1(\underline{x}), \ldots, r_m(\underline{x}))^T$ is the vector of residuals. Assume $r_i(\underline{x})$ are nonlinear.

Differentiating (1) we obtain

$$\nabla_{\underline{x}} f(\underline{x}) = \sum_{i=1}^{m} 2r_i(\underline{x})\nabla_{\underline{x}} r_i(\underline{x}).$$

Let $\mathbf{J}_{\underline{x}}\underline{r}(\underline{x}) = \left\{\frac{\partial r_i}{\partial x_k}\right\}_{ik}$ the Jacobian of $\underline{r}$ at $\underline{x}$: we indicate $\mathbf{J}_{\underline{x}}\underline{r}(\underline{x})$ with $\mathbf{J}(\underline{x})$. Observe $\mathbf{J}(\underline{x}) = \begin{pmatrix} \nabla_{\underline{x}}^T r_1(\underline{x}) \\ \vdots \\ \nabla_{\underline{x}}^T r_m(\underline{x}) \end{pmatrix}$. We can write the expression as:

$$\nabla f(\underline{x}) = 2\mathbf{J}_{\underline{x}}\underline{r}(\underline{x})^T\underline{r}(\underline{x}).$$

The Hessian of f at $\underline{x}$, that we indicate with $\mathbf{H}(\underline{x})$, is

$$\mathbf{H}(\underline{x}) = 2\mathbf{J}(\underline{x})^T\mathbf{J}(\underline{x}) + 2\sum_{i=1}^{m}(r_i(\underline{x}))\nabla^2 r_i(\underline{x}). \tag{2}$$

If the residuals are small, we can discard the last term in (2), leading to

$$\mathbf{H}(\underline{x}) \approx 2\mathbf{J}(\underline{x})^T\mathbf{J}(\underline{x}).$$

Observe that, if the residuals are linear in $\underline{x}$, then $\nabla^2 r_i(\underline{x}) = 0$ for any i, so the second-order approximation is exact and the method is the same as the one for linear least squares.

The Netwon method step:

$$\underline{x}_{k+1} = \underline{x}_k - [\mathbf{H}(\underline{x}_k)]^{-1}\nabla f(\underline{x}_k)$$

corresponds, using approximation of the Hessian to the first derivative, to:

$$\underline{x}_{k+1} = \underline{x}_k - \left[\mathbf{J}(\underline{x}_k)^T\mathbf{J}(\underline{x}_k)\right]^{-1}\mathbf{J}(\underline{x}_k)^T\underline{r}(\underline{x}_k).$$

For the convergence of Gauss-Newton, the inital solution must be sufficiently close to a stationary point of f and the discarded terms (2) must be small. Observe that the method does not need to compute the Hessian $\mathbf{H}(\underline{x})$, that would imply computing the Hessian $\nabla^2 r_i$ for each r_i.

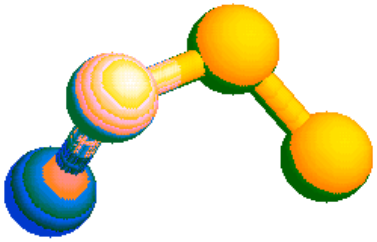


Figure 1: Optimal configuration.

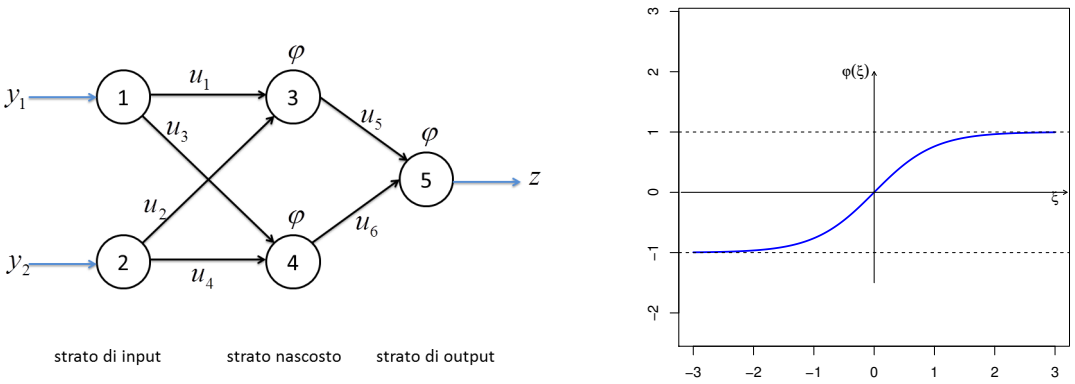
Esercize 2 (extra): neural networks

We have a dataset reported in Table 1, where y_1, y_2 are the input values and z the output of an unknown function: Suppose we want to build a neural network replicating the logical function that generated the data.

a) Let $\varphi(\xi) = \frac{e^{\xi}-e^{-\xi}}{e^{\xi}+e^{-\xi}}$ the activation function, depicted in Figure 2b. Train the network in Figure 2a with the reported dataset (*training set*), finding a set of weights $\underline{u}$ corresponding to the network arcs so to approximate the data. Use the multistart algorithm to find a good local optimum.

Table 1: Exercise 2.

y_1	y_2	z
0.1	0.1	0.1
0.9	0.9	0.05
0.0	0.95	0.98
0.95	0.1	0.95



(a) Neural network considered in the exercise.

(b) Plot of $\varphi(\xi)$.

Figure 2: Features of the neural network.

Use the file `f_hmap.m`, that represents the function implemented by the neural network in Figure 2a:

```
function f = f_hmap(u, y)
     f = f_act(u(5)* f_act(u(1)*y(1) + u(2)*y(2)) ...
             + u(6)* f_act(u(3)*y(1) + u(4)*y(2)));
end

function f = f_act(xi)
  f = (exp(xi) - exp(-xi)) / (exp(xi) + exp(-xi));
end
```

and the file `f_nnet.m`, that encodes the error function to minimize:

```
function f = f_nnet(u)
  y = [ 0.1  0.1  ;
        0.9  0.9  ;
        0.0  0.95 ;
        0.95 0.1  ];
  z = [ 0.1  0.05  0.98  0.95 ]';
  f = 0;
  for i = 1:length(z)
    f = f + ( z(i) - f_hmap(u, [y(i,1) y(i,2)]') )^2;
  end
end
```

b) Compute the outputs with the inputs (0,0), (0,1), (1,0), (1,1) and establish what kind of circuit we have.