```
% to "guess" the underlying mean and standard deviation of data

% r = the ranks of the ordered data (the first=earliest=smallest=highest rank)
% n = total number of data points
% observed = the observed values of our data as they come in
% mu_best = best guess of mean (initiated at most recent data point)
% sigma_best = best guess of std (initated at value 1)
% expectedNormalOrderArray(r,n,mu,sigma)
% = returns array of expected values of ranks r;
% given normal distribution with mean mu and std sigma

r = [100;99;98;97];
n = 100;
observed = [15; 20; 22; 23];

sigma_best = 1;
mu_best = observed(end);

% tweaking some options for fmincon
sigma_options = optimoptions(@fmincon,'StepTolerance', 0.01);

for i = 1: 30 % number of iterations is..arbitrary

    mu_old = mu_best;
    sigma_old = sigma_best;

    % find mu_best with least squares; using sigma_best from previous iteration
    syms mu
    f = sum((expectedNormalOrderArray(r, n, 0, sigma_best) + mu - observed).^2);
    % differentiable: differentiate find min point of sum squares
    mu_best = double(solve(diff(f)==0));

    % round up to speed up
    if round(mu_best) > mu_best
        mu_best = round(mu_best);
    end

    % using mu_best, now find best std (least squares)
    func_min_sigma = @(sigma)sum((expectedNormalOrderArray(r, n, 0, sigma) +↙
mu_best - observed).^2);
    % not differentiable: use fmincon to optimise (this step takes the longest↙
time; ~ 5 seconds)
    sigma_best = fmincon(func_min_sigma, sigma_best,[],[],[],[],[],[],[],↙
sigma_options);


    if abs(mu_best - mu_old) < 0.5 && abs(sigma_best - sigma_old) < 0.05
        break
    end

end
```