Problem 2

The following is the separable unconstrained optimisation problem to be solved:

$$\min_{x,y,z} \qquad f_1(x,z) + f_2(y,z),$$

where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, $z \in \mathbb{R}$ and f_1 and f_2 are quadratic in $n+1$ and $m+1$ variables respectively. In other words, the function f_1 has the form $f_1 = \sum_{i=1}^{n}\sum_{j=i}^{n} a_{ij}x_ix_j + \sum_{i=1}^{n} b_ix_i + c + \sum_{i=1}^{n} d_izx_i + ez + fz^2$, for some coefficients $a_{ij}, b_i, c, d_i, e, f$. The function f_2 can be similarly defined, $f_2 = \sum_{i=1}^{m}\sum_{j=i}^{m} p_{ij}x_ix_j + \sum_{i=1}^{m} q_ix_i + r + \sum_{i=1}^{m} s_izx_i + tz + uz^2$, for some coefficients $p_{ij}, q_i, r, s_i, t, u$.

Once again, z is a shared variable that can be accessed by all agents, while x and y are both vectors of private variables that can only be accessed by Agent 1 and Agent 2 respectively.

By applying a change of variables, the cost function can be separated and the new formulation of the problem will be

$$\begin{aligned} \min_{x,\xi_1,y,\xi_2} \qquad & f_1(x,\xi_1) + f_2(y,\xi_2) \\ s.t. \qquad & \xi_1 = \xi_2. \end{aligned}$$

The dual function will then be

$$\begin{aligned} q(\lambda) &= q_1(\lambda) + q_2(\lambda) \\ &= \inf_{x,\xi_1}[f_1(x,\xi_1) - \lambda\xi_1] + \inf_{y,\xi_2}[f_2(y,\xi_2) + \lambda\xi_2]. \end{aligned}$$

The minimisers can be found by finding the gradients and setting to zero, as follows:

$$\begin{aligned} \nabla[f_1(x,\xi_1) - \lambda\xi_1] &= \begin{bmatrix} 2a_{11}x_1 + \sum_{i\neq 1} a_{1i}x_i + b_1 + d_1\xi_1 \\ \vdots \\ 2a_{nn}x_n + \sum_{i\neq n} a_{ni}x_n + b_n + d_n\xi_1 \\ 2f\xi_1 + (e-\lambda) + \sum_{i=1}^{n} d_ix_i \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} 2a_{11} & \dots & a_{1n} & d_1 \\ & \vdots & & \\ a_{1n} & \dots & 2a_{nn} & d_n \\ d_1 & \dots & d_n & 2f \end{bmatrix}}_{A_1} \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ \xi_1 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1 \\ \vdots \\ b_n \\ (e-\lambda) \end{bmatrix}}_{B_1} = \mathbf{0} \end{aligned}$$

$$\nabla[f_2(y,\xi_2)+\lambda\xi_2] = \begin{bmatrix} 2u\xi_2 + (t+\lambda) + \sum_{i=1}^{m} s_i y_i \\ 2p_{11}y_1 + \sum_{i\neq 1} p_{1i}y_i + q_1 + s_1\xi_2 \\ \vdots \\ 2p_{mm}x_m + \sum_{i\neq m} q_{mi}y_m + q_m + s_m\xi_2 \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} 2u & s_1 & ... & s_m \\ s_1 & 2p_{11} & ... & p_{1m} \\ & & \vdots & \\ s_m & p_{1m} & ... & 2p_{mm} \end{bmatrix}}_{A_1} \begin{bmatrix} \xi_2 \\ y_1 \\ \vdots \\ y_m \end{bmatrix} + \underbrace{\begin{bmatrix} (t+\lambda) \\ q_1 \\ \vdots \\ q_m \end{bmatrix}}_{B_1} = \mathbf{0}$$

The minimiser of $[f_1(x,\xi_1)-\lambda\xi_1]$ is then the solution to the set of $n+1$ linear equations and the minimiser of $[f_2(y,\xi_2)+\lambda\xi_2]$ is the solution to the set of $m+1$ linear equations above. Note, the matrices A_1 and A_2 are symmetric and must be invertible, i.e., their eigenvalues must be strictly positive; the problem is ill-conditioned if the eigenvalues are close to zero.

The same method of using the subgradient and dual decomposition from Problem 1 now applies to this problem.

Combined Problem

At this stage it is useful to observe the problem without separation, where the cost function is optimised as a single function. The problem is quadratic, and the minimiser is found by solving $n+m+1$ linear equations. This means solving the problem $Ax = B$, with A and B found by augmenting and overlapping A_1, A_2 and B_1, B_2 respectively. This is visualised below.

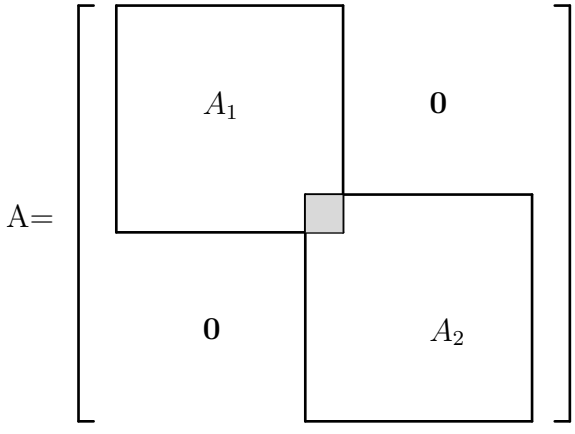


Figure 1: Overlapping squares represent how A_1 and A_2 overlap. There is a single element in the grey overlapping region and it is equal to $2f+2u$, the sum of the elements of A_1 and A_2 that overlap.

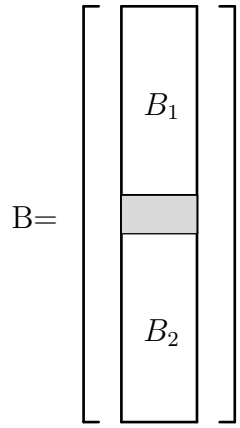


Figure 2: Similarly, B is a vector with B_1 and B_2 overlapping as shown. Only one element is in the grey overlapping region, and that element is equal to $e + t$, the sum of the elements of B_1 and B_2 that overlap.

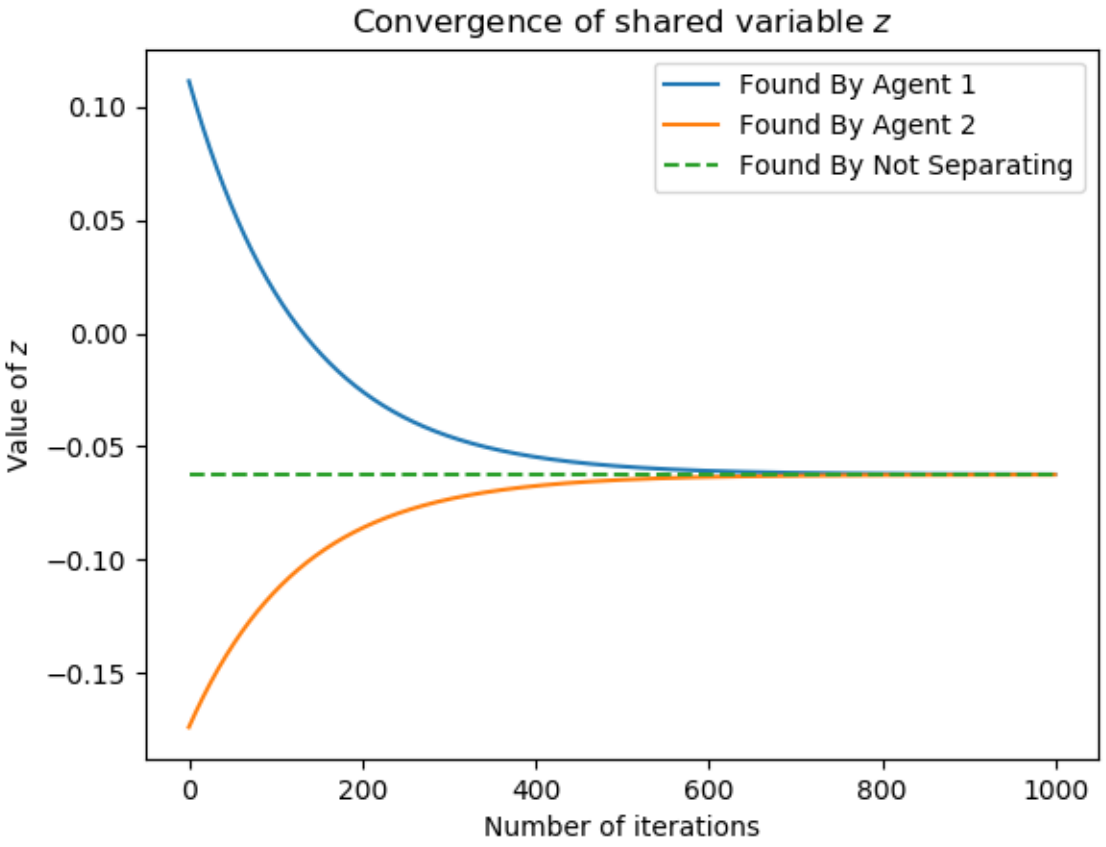
We can solve the combined problem and use it to check the convergence of the decomposed problem.

Convergence

As a reminder, in separating the cost function, it is now a requirement that both A_1 and A_2 are well-conditioned in order to get convergence. The combined matrix A may be well conditioned, but depending on how A_1 and A_2 are separated, we may end up with an ill-conditioned matrices.

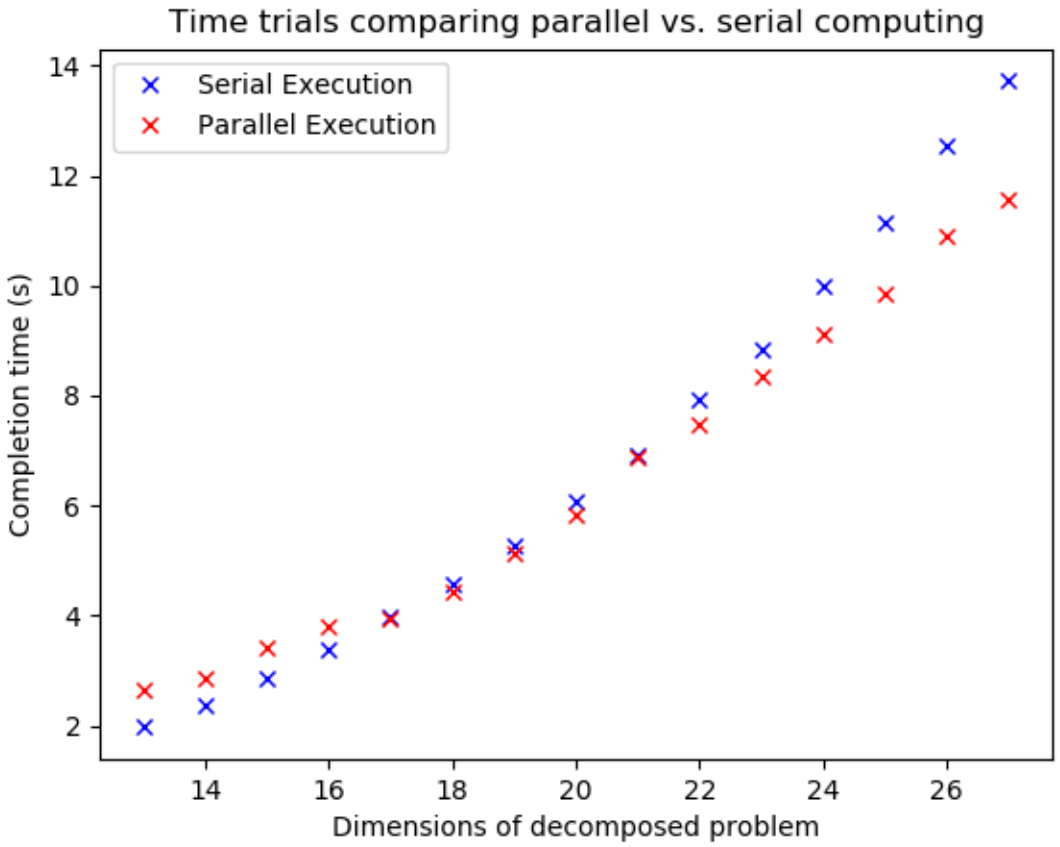
This picks up from where Problem 1 left off. In Problem 1, convergence was still guaranteed regardless of how the shared variable x_3^2 was separated. It turns out that this does not hold for coefficients of x_3^2 that are too close to zero. If the coefficient of x_3^2 is too small, convergence no longer occurs. Thus, we see the key requirement that the matrices are well-conditioned appearing in Problem 1 as well.

When the problem is well-conditioned, the separated problem when the dual decomposition with subgradient method is used converges to the solution obtained by solving the combined problem.



Computational Overhead

There is some computational overhead to spawn a Process on Python. Previously in Problem 1, a dummy for-loop was added to increase the complexity of each parallel process. For Problem 2, we achieve the same affect by increasing n and m, i.e., increasing the dimensions of A_1 and A_2. The following plot shows the completion times, where the dimensions of A_1 and A_2 have been set to be equal.



In Problem 1, the increase in completion times for both serial and parallel were linear. This was because a single dummy for-loop was used to simulate increasing the complexity of the processes. In Problem 2, the solution to $A_1x + B_1$ and $A_2x + B_2$ were found by applying Gaussian elemination, which has a complexity of order $\mathcal{O}(n^3)$. The sample plot above may not be enough to conclude cubic growth as the dimensions of the problem grow, but it can be seen that the growth is of order higher than linear growth.

How to Run

The above examples

Make sure you are in the correct directory. Then to run the test that generated the above plots, execute the **main.py** file, i.e. use the command

>>>**python main.py**

Function Descriptions

The function **parallel.do_parallel**, description.

Syntax: do_parallel(max_iter,alpha,A1,A2,b1,b2,verbose=False)

Parameter values:

- max_iter, Required. Number of iterations for the subgradient method.
- alpha, Required. Step size for the subgradient method.
- A1, Required. The matrix of coefficients A_1 as described above.
- A2, Required. The matrix of coefficients A_2 as described above.
- b1, Required. The matrix of coefficients B_1 as described above.
- b2, Required. The matrix of coefficients B_2 as described above.
- verbose, Default False. Print results to screen.

Outputs:

- Output 1. List containing ξ_1^* for all iterations of the subgradient method.
- Output 2. List containing ξ_2^* for all iterations of the subgradient method.
- Output 3. Completion time.