Point to Point: Exercise 1

p processes participate in a computation; they are logically ordered as a ring.
Process P_i, (with $i \in [0, \ldots, p-1]$), owns a local data buffer v_i containing n integers.
In order to make progress, P_i needs to compute

$$v_i^{\text{next}} := f(v_{i-1}^{\text{now}}) + f(v_i^{\text{now}}) - v_{i+1}^{\text{now}}.$$

The function $f : \mathbb{R}^n \to \mathbb{R}^n$ is expensive and does not overwrite its argument.
The buffers v_i^{now} and v_i^{next} are distinct.

Write a program that performs one step of the computation (from v_i^{now} to v_i^{next}), aiming to minimize the execution time. Add short comments explaining the ideas.

Point to Point: Exercise 2

Three processes participate in a computation:
p_0, p_1, and p_2 own the square matrices A, B, and C, respectively.
All matrices are of size $n \times n$.
Each process has enough memory space to store 4 matrices.
The functions f and g are sequential and expensive; their execution takes much longer than the time necessary to add two matrices; g is more expensive than f. The function f overwrites its input; g doesn't.
Process p_0 has to compute $X := f(A) + B + C$.
Process p_1 has to compute $Y := A + g(B) - C$.
Process p_2 has to compute $Z := f(A) + g(C)$.

Write pseudocode for p_0, p_1, and p_2, mimicking MPI's constructs (avoid ambiguity). For example, if clear, you can use `Ssend( Buf, 2 );` in place of `MPI_Ssend( Buf, size, type, 2, tag, comm );`

Minimize the execution time. Explain your decisions.

```
int f( double *M, int n );
int g( double *M, int n, double *Out );

int main( int argc, char *argv[] ){
   int i, me, nProcs, size;
   double time;

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &me );
   MPI_Comm_size( MPI_COMM_WORLD, &nProcs );

   srand48( (me+1) * (unsigned)time( (time_t *)NULL) );
   size = // something; nothing to do here

   // -- your pseudocode --

   // initializations


   MPI_Barrier( MPI_COMM_WORLD );
   time = MPI_Wtime( );

   // -- your pseudocode --

   // use a separate sheet


   MPI_Barrier( MPI_COMM_WORLD );
   time = MPI_Wtime() - time;

   MPI_Finalize();
   free( Sol );
   return 0;
}
```