Test sample (Minimizing energy of 1D chain of balls connected by spring)

In this problem we want to minimize the following energy equation for 1D chain of balls connected by spring

$$E = \sum_{i=0}^{N-1} \left((X_{i+1} - X_i) - L \right)^2$$

The L is the ideal distance between 2 points. The balls have been initially placed with different distance W.

$$X_i = i \times W$$

And the final solution would be

$$X_i = i \times L + c$$

Where c is a constant since we didn't fix the boundaries, it is possible to have a constant shift.

We will use SNES to solve the problem recalling that in relaxed system

$$\frac{dE}{dX_i} = F(\{X_i\}) = 0$$

So the function in SNES would be $\frac{dE}{dX_i}$ and Jacobian $\frac{d^2E}{dX_i^2}$.

$$F_{SNES} = \frac{dE}{dX_i} = \begin{cases} -2(X_{i+1} - X_i - L) & i = 0 \\ 2(2X_i - X_{i-1} - X_{i+1}) & 0 < i < N-1 \\ +2(X_i - X_{i-1} - L) & i = N-1 \end{cases}$$

F_{SNES} implementation in the PETSc code

```
ierr = VecGetArray(x,&xx);CHKERRQ(ierr);
ierr = VecGetArray(f,&ff);CHKERRQ(ierr);

/* Compute function */
ff[0] = -2.0*(xx[1]-xx[0]-2.0);
for(i=1; i<N-1; i++)
{
       ff[i] = 2*(2*xx[i]-xx[i-1]-xx[i+1]);
       printf("x[%d]=%f \n",i,xx[i]);
}
ff[N-1] = +2*(xx[N-1]-xx[N-2]-2);
/* Restore vectors */
ierr = VecRestoreArray(x,&xx);CHKERRQ(ierr);
ierr = VecRestoreArray(f,&ff);CHKERRQ(ierr);
```

And the Jacobian would be

$$J = \frac{d^2E}{dX_i^2} = \begin{bmatrix} 4 & -2 & & \\ -2 & 4 & -2 & \\ & -2 & 4 & \ddots \\ & & \ddots & \ddots \end{bmatrix}$$

$Jacobian_{SNES}$ implementation in the PETSc code

```
// Set the left corner
i = 0; // First Row
j[0] = 0;           A[0] = +4;
j[1] = 1;           A[1] = -2;
ierr = MatSetValues(*B,1,&i,2,j,A,INSERT_VALUES);CHKERRQ(ierr);
ierr = MatSetValues(*jac,1,&i,2,j,A,INSERT_VALUES);CHKERRQ(ierr);

for(i=1; i<N-1; i++)
{
       printf("Jac: x[%d]=%f \n",i,xx[i]);
       j[0] = i-1;   A[0] = -2;
       j[1] = i;           A[1] = +4;
       j[2] = i+1;   A[2] = -2;
       ierr = MatSetValues(*B,1,&i,3,j,A,INSERT_VALUES);CHKERRQ(ierr);
       ierr = MatSetValues(*jac,1,&i,3,j,A,INSERT_VALUES);CHKERRQ(ierr);
}

// Set the right corner
i = N-1;      // Last Row
j[0] = i-1;         A[0] = -2;
j[1] = i;                 A[1] = +4;
ierr = MatSetValues(*B,1,&i,2,j,A,INSERT_VALUES);CHKERRQ(ierr);
ierr = MatSetValues(*jac,1,&i,2,j,A,INSERT_VALUES);CHKERRQ(ierr);
```

The initialization has been done according to

$$X_i = i \times W$$

$Initialization$ implementation in the PETSc code

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Evaluate initial guess; then solve nonlinear system
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
ierr = VecGetArray(x,&xx);CHKERRQ(ierr);
for(i=0; i<=N-1; i++)
{
       xx[i] = Initial_Spacing * (double) i;
}
ierr = VecRestoreArray(x,&xx);CHKERRQ(ierr);
```

Allocating memory for Jacobian matrix and function vector and solving SNES

```
        /*
        Create vectors for solution and nonlinear function
        */
        ierr = VecCreate(PETSC_COMM_WORLD,&x);CHKERRQ(ierr);
        ierr = VecSetSizes(x,PETSC_DECIDE,N);CHKERRQ(ierr);
        ierr = VecSetFromOptions(x);CHKERRQ(ierr);
        ierr = VecDuplicate(x,&r);CHKERRQ(ierr);

        /*
        Create Jacobian matrix data structure
        */
        ierr = MatCreate(PETSC_COMM_WORLD,&J);CHKERRQ(ierr);
        ierr = MatSetSizes(J,PETSC_DECIDE,PETSC_DECIDE,N,N);CHKERRQ(ierr);
        ierr = MatSetFromOptions(J);CHKERRQ(ierr);

        /*
        Set function evaluation routine and vector.
        */
        ierr = SNESSetFunction(snes,r,FormFunction1,&user);CHKERRQ(ierr);

        /*
        Set Jacobian matrix data structure and Jacobian evaluation routine
        */
        ierr = SNESSetJacobian(snes,J,J,FormJacobian1,PETSC_NULL);CHKERRQ(ierr);

        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        Customize nonlinear solver; set runtime options
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        /*
        Set linear solver defaults for this problem. By extracting the
        KSP, KSP, and PC contexts from the SNES context, we can then
        directly call any KSP, KSP, and PC routines to set various options.
        */
        ierr = SNESGetKSP(snes,&ksp);CHKERRQ(ierr);
        ierr = KSPGetPC(ksp,&pc);CHKERRQ(ierr);
        ierr = PCSetType(pc,PCNONE);CHKERRQ(ierr);
        ierr = SNESSetTolerances(snes,1.e-3,1.e-6,1.e-6,1000,10000);CHKERRQ(ierr);

        /*
        Set SNES/KSP/KSP/PC runtime options, e.g.,
        -snes_view -snes_monitor -ksp_type <ksp> -pc_type <pc>
        These options will override those specified above as long as
        SNESSetFromOptions() is called _after_ any other customization
        routines.
        */
        ierr = SNESSetFromOptions(snes);CHKERRQ(ierr);

        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        Evaluate initial guess; then solve nonlinear system
        - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        ierr = VecGetArray(x,&xx);CHKERRQ(ierr);
        for(i=0; i<=N-1; i++)
        {
                xx[i] = Initial_Spacing * (double) i;
        }
        ierr = VecRestoreArray(x,&xx);CHKERRQ(ierr);
```

```
      /*
      Note: The user should initialize the vector, x, with the initial guess
      for the nonlinear solver prior to calling SNESSolve().  In particular,
      to employ an initial guess of zero, the user should explicitly set
      this vector to zero by calling VecSet().
      */

      ierr = SNESSolve(snes,PETSC_NULL,x);CHKERRQ(ierr);
      ierr = SNESGetIterationNumber(snes,&its);CHKERRQ(ierr);
```