EXCERCISE: ELEMENT-BY-ELEMENT APPROXIMATIONS

SIVAN TOLEDO AND GIL SHKLARSKI

In this exercise we will develop code that takes a finite-elements matrix in which the null space of all the element matrices is the constant vector and approximates it by another finite-elements matrix in which the element matrices are all weakly diagonally dominant.

Our input is a matrix K given as an explicit sum $K = \sum_e K_e$, where each K_e is symmetric positive semidefinite with only 3 nonzero rows and columns. The rank of every K_e is exactly 2 and the constant vector is in its null space. The elements represent a discretization of physical problem. You generate the K_e's using the MATLAB command

```
>> m = generate_simple_model(1000,'poisson',true);
```

The number 1000 is a target number of unknowns in the discretization; the actual number will vary a bit. If you pass `true` in the last argument, you will see the geometry of the two-dimensional finite-elements mesh that was used. The function `generate_simple_model` returns a structure with three members: the number of unknowns, the number of elements, and an array of the K_e's. Each K_e is represented by a dense 3-by-3 matrix and an array of 3 global row/column indices.

```
>> m
m =
    total_num_vars: 1008
        n_elements: 1860
          elements: [1860x1 struct]
>> m.elements(1)
ans =
            matrix: [3x3 double]
    local_to_global: [1 17 2]
```

(1) Write a MATLAB function `assemble(m)` that takes the representation of K that `generate_simple_model` returns and returns an explicit sparse matrix K. Verify that K has the right size and the correct rank and null space (the constant vector). This function can be implemented efficiently, but it is enough to just make it work. Debug your code on small problems.

```
>> m = generate_simple_model(1000,'poisson',true);
>> A = assemble(m);
```

```
>> x = rand(1008,1);
>> b = A*x;
>> m2 = approximate_model(m);
>> B = assemble(m2);
>> % make B nonsigular by enforcing z(1)=0 on Bz=r
>>
```

(2) Generate a random solution vector and a right-hand side, and try to solve the problem using MINRES or preconditioned Conjugate Gradients with no preconditioning. Try to get a sense for the scaling of the solver (performance as a function of problem size). You can either solve the singular system or force one unknown to zero by adding 1 to one diagonal element in K.

```
>> Kes = generate_simple_model(1000,'poisson',true);
>> K = assemble(Kes);
>> x = rand(1008,1);
>> b = K*x;
...
>> xhat = minres(K,b,1e-8,1000);
...
```

(3) Now write a function that approximates a single dense matrix $\tilde{K}_e$ by a symmetric and weakly diagonally dominant matrix $\tilde{L}_e$. Assume that $\tilde{K}_e$ is positive semidefinite and that its null space is spanned by the constant vector. Scale $\tilde{L}_e$to ensure that $\lambda(\tilde{K}_e, \tilde{L}_e) \geq 1$.
   (a) Generate the incidence matrix V of the clique (whose size is the order of $\tilde{K}_e$).
   (b) Decompose $\tilde{K}_e = UU^T$. You can use the eigendecomposition of $\tilde{K}_e$.
   (c) Compute U^+V (you can compute U^+ directly from the eigendecomposition of $\tilde{K}_e$).
   (d) Compute the diagonal matrix D that scales the columns of U^+VD to unit 2-norm.
   (e) Compute a weakly diagonally-dominant approximation VD^2V^T.
   (f) Scale VD^2V^T to ensure $\lambda(\tilde{K}_e, \alpha_e VD^2V^T) \geq 1$ and return $\alpha_e VD^2V^T$.

(4) Implement a wrapper `approximate_model` that takes the element-by-element representation of K and returns an approximation $L = \sum_e L_e$ in which every L_e is a weakly diagonally-dominant approximation of K_e. The distinction betweenK_e and $\tilde{K}_e$ (and similarly for L_e and $\tilde{L}_e$) is that the first is a large sparse matrix and the second is just the nonzero rows and columns of it. Assemble L and verify that the order and null space of L are correct.

```
>> Les = approximate_model(Kes);
>> L = assemble(Les);
...
```

(5) Modify L to ensure that it is nonsingular and use it as a preconditioner. How is the convergence related to the problem size?

```
>> xhat = minres(K,b,1e-8,1000,L);
...
```

(6) Use the Splitting Lemma to bound $\Lambda(K, L)$ from above and below (you will need to explicitly compute bounds on $\Lambda(K_e, L_e)$, but the computation of α_e should yield them for free). Compare the bound to the actual extreme eigenvalues of (K, L). Is it tight? (Computing the exact eigenvalues is expensive for large K's, so start from small ones.)

```
>> e = sort(eig(pinv(full(L))*full(K)));
>> [min(e(2:end)) max(e(2:end))]
```