Project 2

Problem

In this project you will sove a 2D advection-diffusion equation given by

$$-\Delta u + \mathbf{v} \cdot \nabla u = f, \qquad 0 \leq x, y \leq 1 \tag{1a}$$

$$u = g \qquad \text{on the boundary} \tag{1b}$$

using GMRES and multigrid. For simplicity, we assume that the velocity field $\mathbf{v} = [v_1, v_2]^\top \geq 0$, in the sense that both components are non-negative.

We consider here a finite differences discretization on equidistant nodes with the 5-points formula for the diffusion term and an upwind scheme for the advection term. The step-size is $h = 1/N$ in both the x- and the y- direction. In the discrete problem, the function f is known and the function u is sought at grid points $x_i = ih$ and $y_j = jh$ and on these points they have values $\{f_{i,j}\}_{i,j=0}^N$, and $\{u_{i,j}\}_{i,j=0}^N$ respectively. The discrete solution becomes

$$(4U_{i,j} - U_{i+1,j} - U_{i-1,j} - U_{i,j+1} - U_{i,j-1}) + hv_1(U_{i,j} - U_{i-1,j}) + hv_2(U_{i,j} - U_{i,j-1}) = h^2 f_{ij} \tag{2}$$

Notice that this discretization results in a diagonal dominant matrix, independent of h for $\mathbf{v} \geq 0$.

All routines should work directly on the grid. This means that you should not store the matrix of coefficients of the linear system, but implement the effect of applying it directly on a function known on the nodes of the grid as in (1). The right-hand side, the numerical solution, the residual and the error should be considered as functions whose values are known on the grid, and stored as $(N+1) \times (N+1)$ matrices (including the boundary points).

Throughout the project, let the velocity field $\mathbf{v} = [1, 1]^\top$ and choose the right hand side function and the boundaries such that the exact solution of the PDE (1) is

$$u(x, y) = \sin(\pi x) \sin(2\pi y).$$

Tasks

0) Write a function taking an $(N+1) \times (N+1)$ array U as input, and return the discrete diffusion-advection operation described by (2). Then use `scipy.sparse.gmres` with tight tolerances to compute solution of the discrete problem for different values of N. These will later be used as reference solutions.

1) Implement a version of the restarted GMRES (or FOM) method working directly on the grid. The code should contain a convergence test so that it terminates when

$$\frac{\|r_k\|_2}{\|r_0\|_2} < \text{tol}.$$

2) Create a function implementing a multigrid V-cycle for the problem (1). As input the function should take the initial guess, the right-hand side, the velocity field, the size of the finest grid, the number of pre-smoothings, the number of post-smoothings, the current level, and the level corresponding to the coarsest grid. Use the GMRES algorithm to solve the problem on the coarsest level. As a relaxation method, use the weighted Jacobi iterations with $\omega = 2/3$. Use linear interpolation and full-weighting to transfer information between the grids.

   It is an advantage to break the code down into several functions. The main routine could for instance look as follows:

```
def mgv(u0,rhs,v,N,nu1,nu2,level,max_level):
"""
    the function mgv(u0,rhs,N,nu1,nu2,level,max_level) performs
    one multigrid V-cycle on the 2D advection-diffusion equation on the unit
    square [0,1]x[0,1] with initial guess u0 and righthand side rhs.

    input:  u0        - initial guess
            rhs       - righthand side
            v         - velocity field
            N         - u0 is a (N+1)x(N+1) matrix
            nu1       - number of presmoothings
            nu2       - number of postsmoothings
            level     - current level
            max_level - total number of levels
"""

    if level==max_level:
        u,resvec,iter = gmres(u0,rhs, .... )
    else:
        u = jacobi(u0,rhs,2/3,N,nu1)
        rf = residual(u,rhs,N)
        rc = restriction(rf,N)
        ec = mgv(np.zeros_like(rc),rc,v,int(N/2),nu1,nu2,level+1,max_level)
        ef = interpolation(ec, int(N/2))
        u = u + ef
        u = jacobi(u,rhs,2/3,N,nu2)
    return u
```

Notice that the routine is defined recursively. What is left is to implement the functions jacobi, residual, restriction, interpolation.

Use a random initial guess for the interior points (NumPy function numpy.random.rand()). Test the program for different choices of the number of pre-and post-smoothings. How does the number of iterations to convergence depends on the size of the problem?

3) Use your codes to solve an advection-diffusion equation of your own choice. What happens e.g. if the velocity field is more dominating? Can you extend your code to a space dependent velocity field (but it should be positive)?

4) Modify the routine in 2) to solve the diffusion-advection problem with a preconditioned GMRES method using the multigrid V-cycle from 2) as a preconditioner. The application of a preconditioner M implies that we have to solve the linear system

$$Mz = r$$

once for each GMRES iteration. Using the multigrid V-cycle as preconditioner corresponds to setting z equal to the result of one V-cycle with zero initial guess and the residual as righthand side, i.e.

```
z = mgv(np.zeros((N+1,N+1)),r,N,nu1,nu2,1,max_level);
```

Test the algorithm on the same problem as in 2). How does the number of iterations to reach convergence depend on the size of the problem?

Code advice

Use matrix- or vector-operations whenever possible. For example for the 5-point stencil with advection terms, consider the discretized advection-diffusion equation:

$$-\Delta u + \mathbf{v} \cdot \nabla u = f$$

where $\mathbf{v} = (v_1, v_2)$ is the velocity field. Using upwind differencing for the advection terms and central differences for the diffusion terms, the Jacobi iteration on component form is given by:

$$u_{i,j}^{(k+1)} = \frac{u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)}(1 + v_1 h) + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)}(1 + v_2 h) + h^2 f_{i,j}}{4 + v_1 h + v_2 h}, \quad 1 \leq i, j \leq N - 1$$

Using numpy.ix_ of NumPy to define appropriate arrays of indices, this can be written as a matrix-operation as follows:

```
index = np.arange(1,N)  # indices corresponding to internal nodes
ixy = np.ix_(index,index)
ixm_y = np.ix_(index-1,index)
ixp_y = np.ix_(index+1,index)
ix_ym = np.ix_(index,index-1)
ix_yp = np.ix_(index,index+1)

# Denominator includes advection terms
div = (4 + v[0]*h + v[1]*h)

# Jacobi iteration with upwind differencing for advection
u[ixy] = (u[ixp_y] + u[ixm_y]*(1 + v[0]*h) +
          u[ix_yp] + u[ix_ym]*(1 + v[1]*h))/div + (h**2/div)*f[ixy]
```

which is much more efficient than the component-wise implementation.

The vectorized implementation not only provides better performance but also leads to clearer, more maintainable code. Note that the advection terms introduce an asymmetry in the discretization through the upwind differencing, which is reflected in the $(1 + v_i h)$ terms multiplying the upstream values.

Try to make simple tests to check that the different sub-routines or functions work as expected. For example, make a test of the restriction and interpolation routines by applying them to a very simple matrix and check that the result is as expected. This approach makes debugging much easier.

Report

Write your answers in a Jupyter notebook and include a short introduction. Put the main functions into **separate .py-files** that you import into the notebook.

The project asks you to implement three different methods: GMRES, multigrid V-cycle, and a preconditioned GMRES method using the multigrid V-cycle as a preconditioner.

For each method, report the number of iterations to convergence and computational time (using i.e time.perf_counter()) for different N values. Choose for instance $N = 2^5, 2^6, 2^7, 2^8$. Additionally, for one specific choice of N, plot the solution and the convergence history by plotting the 2-norm of the residual against the number of iterations. Use a logarithmic scale on the y-axis with for instance semilogy of matplotlib. Use

$$\frac{\|r_k\|}{\|r_0\|} < \text{tol}$$

as convergence criteria in all the computations for some small tolerance of your choice.

For all exercises, provide comments on the results. Were the results as expected? If not, why?