Scientific Computing Homework 2

1 Spectral Radius of Fixed Point Methods

Fixed Point iterative methods involve finding the solutions to

$$Ax = b \tag{1}$$

by decomposing the matrix into a form such that iterations are

$$x_{n+1} = Bx_n + c, \tag{2}$$

where B is matrix that is typically a function of A, and c is a function of A and b. These iteration matrices are generally most easily expressed when A is decomposed as its diagonal, lower, and upper triangular components:

$$A = D + L + U, \tag{3}$$

since the forms of these matrices make them easily invertible for use in the iterative updates to x.

Given an initial guess x_0 the error $e_n = x_n - x$ is

$$e_n = B^n e_0, \tag{4}$$

and the relative norm of the error is:

$$\frac{\|e_n\|}{\|e_0\|} \leq \|B^n\|. \tag{5}$$

This norm of $\|B^n\|$ can be approximated by upper bounds, and in the case of the 2-norm, in terms of the spectral radius $\rho(B)$:

$$\|B^n\| \leq \|B\|^n \leq \rho(B)^n. \tag{6}$$

Therefore the spectral radius of B can be found from either directly calculating (through some other iterative method using a numerical library) for the eigenvalues of B, or from measuring the norms of the residual over the iterations, with the last iteration likely giving the best estimate for the spectral radius:

$$\rho(B) = \max_{\lambda} |\lambda(B)| = \left(\frac{\|e_n\|}{\|e_0\|} \right)^{1/n}. \tag{7}$$

The condition number of A is defined in the 2-norm, and can be defined,

$$\kappa(A) = \|A\| \left\|A^{-1}\right\| = \frac{\sigma_{\max}}{\sigma_{\min}} \overset{A^T A = A A^T}{=} \frac{\lambda_{\max}}{\lambda_{\min}}, \tag{8}$$

where σ are the singular values of A, and the last equality for eigenvalues is solely when A is normal.

1.1 Jacobi Method

For the Jacobi method, the iterations proceed such that

$$B = -D^{-1}(L+U) \tag{9}$$
$$c = D^{-1}b. \tag{10}$$

1.2 Gauss-Seidel Method

For the Gauss-Seidel method, the iterations proceed such that

$$B = -(D+L)^{-1}U \tag{11}$$
$$c = (D+L)^{-1}b. \tag{12}$$

1.3 Fixed Point Iteration Results

As shown in Fig. 1, the numerical and analytic results somewhat agree, due to the analytic result of the relative error being an upper bound, and so the analytic spectral radius will generally be larger than the estimate from numerical error of the iterative method. In general, for both the Jacobi and Gauss-Seidel methods, the numerical and analytic methods roughly agree, converging to $\rho \to 1$ (from other analytic analysis).

In general, the spectral radius $\rho(B)$ is smaller for the **Gauss-Seidel** method. This is assumed to be because this method can be shown to converge polynomially faster than the Jacobi method, leading to the error, scaling as ρ^n, decreasing faster if the spectral radius is smaller.

When the condition number is calculated for A, it is seen that due to the condition number being proportional to the reciprocal of the minimum eigenvalue, and that eigenvalue $\lambda_{\min} \to 0$ as the grid size approaches infinity, the condition number will grow without bound.

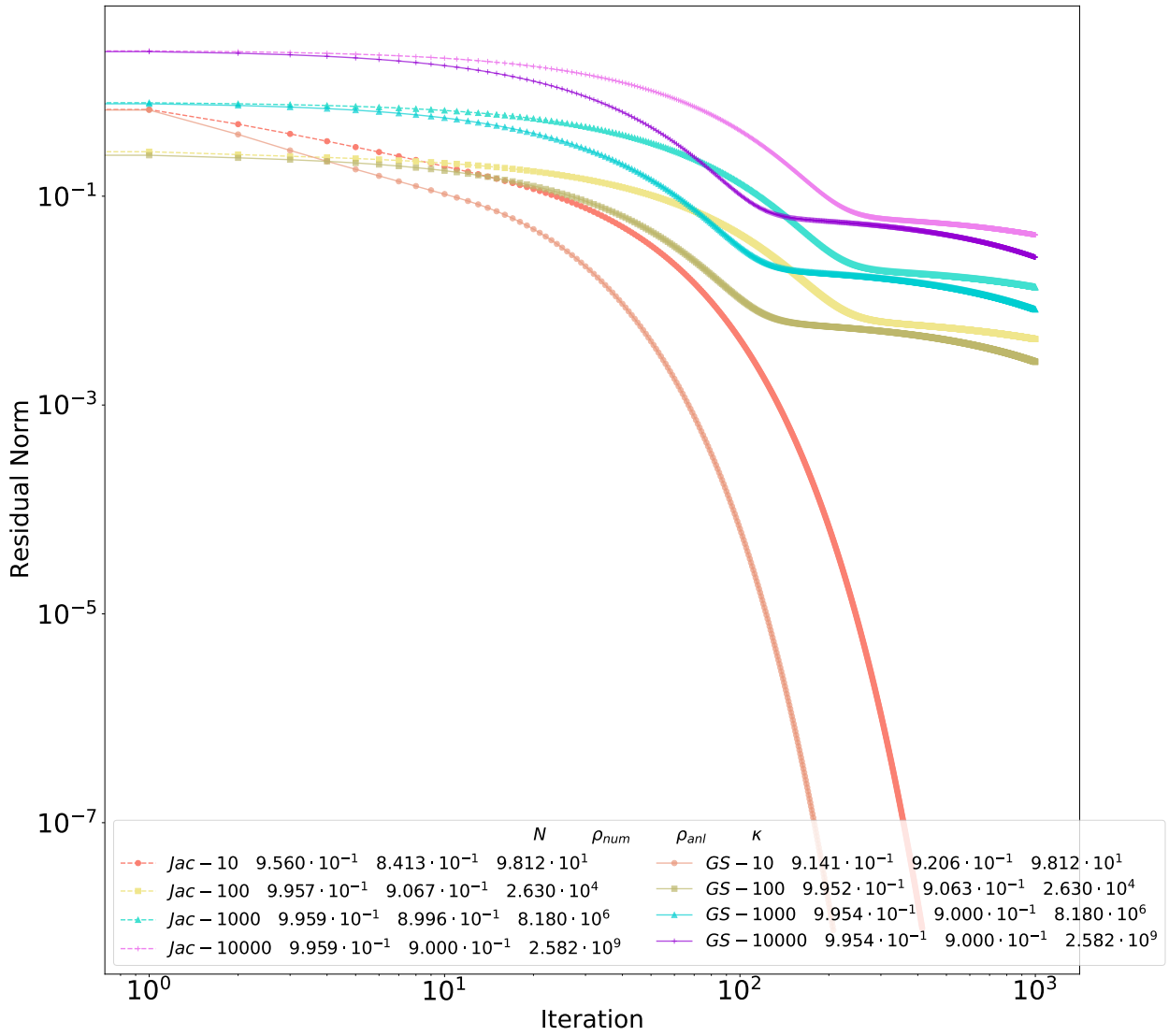

Figure 1: Residual norms over iterations for $d = 1$ Laplace's equation, using Jacobi and Gauss-Seidel methods. Shown in the legend are eigenvalue statistics for various grid sizes.

Please refer to the code in Code. 2 the Appendix that produces these results.

2 QR Factorization

In order to perform the QR factorization, the following QR,class in Code. 3 was used. The QR algorithm uses the modified Graham-Schmidt algorithm, which orthogonalizes a basis for A by removing parallel components from later column vectors for A, before they are transformed into their basis counterpart.

The matrix to be factored is the $2 \times d$-pt stencil for the Laplacian, for $d = 2$ dimensions, with Dirichlet (non-periodic) boundary conditions, for a system with length along each dimension n. Here, the central position x_{ij} is multiplied by $2d$, and subtracted from it are the nearest neighbors in the Cartesian grid, which are some permutations of $\pm 1, \pm q*n$, some multiple of n. This construction of the matrix, and results are from Code. 5.

The code is compiled from the Makefile in Code. 6, and when run ¡n¿ is called, the resulting QR algorithm output is shown in Code. 1. The output includes printouts for $n = 4$, and shows the matrices A, Q, R, as well as the orthonormality of the column vectors q_i of Q, and the product of the QR matrices, yielding back A (within 10^{-15}).

Code 1: QR factorization output

```
2pt stencil Laplacian matrix in d=2
h^2 factor omitted for debugging
values < 1e-15 rounded to 0 for debugging

A
-4   1   0   0   1   0   0   0   0   0   0   0   0   0   0   0
1   -4   1   0   0   1   0   0   0   0   0   0   0   0   0   0
0   1   -4   1   0   0   1   0   0   0   0   0   0   0   0   0
0   0   1   -4   0   0   0   1   0   0   0   0   0   0   0   0
1   0   0   0   -4   1   0   0   1   0   0   0   0   0   0   0
0   1   0   0   1   -4   1   0   0   1   0   0   0   0   0   0
0   0   1   0   0   1   -4   1   0   0   1   0   0   0   0   0
0   0   0   1   0   0   1   -4   0   0   0   1   0   0   0   0
0   0   0   0   1   0   0   0   -4   1   0   0   1   0   0   0
0   0   0   0   0   1   0   0   1   -4   1   0   0   1   0   0
0   0   0   0   0   0   1   0   0   1   -4   1   0   0   1   0
0   0   0   0   0   0   0   1   0   0   1   -4   0   0   0   1
0   0   0   0   0   0   0   0   1   0   0   0   -4   1   0   0
0   0   0   0   0   0   0   0   0   1   0   0   1   -4   1   0
0   0   0   0   0   0   0   0   0   0   1   0   0   1   -4   1
0   0   0   0   0   0   0   0   0   0   0   1   0   0   1   -4

QR Factorization
R
4.24264   -1.88562   0.235702   0   -1.88562   0.471405   0   0   0.235702   0
  0   0   0   0   0   0
0   3.92994   -1.92256   0.254457   -0.395822   -1.80947   0.508913   0
  0.113092   0.254457   0   0   0   0   0   0
0   0   3.90489   -1.92343   -0.0810641   -0.407163   -1.79815   0.512178
  0.0414532   0.125281   0.256089   0   0   0   0   0
```

```
0   0   0   3.77302   -0.0146306   -0.085533   -0.420912   -1.85922   0.0135052
  0.0467055   0.13055   0.26504   0   0   0   0
0   0   0   0   3.90909   -2.01111   0.268481   0.00366266   -1.92046   0.540166
  0.00579921   0.000991971   0.255814   0   0   0
0   0   0   0   0   3.50513   -2.08475   0.301524   -0.499462   -1.82539
  0.606853   0.00703671   0.146776   0.285296   0   0
0   0   0   0   0   0   3.45141   -2.09595   -0.145732   -0.531689   -1.80245
  0.615969   0.0687576   0.172327   0.289737   0
0   0   0   0   0   0   0   3.28589   -0.0438041   -0.165344   -0.562795
  -1.89243   0.0301042   0.0837416   0.184813   0.304332
0   0   0   0   0   0   0   0   3.86898   -2.0661   0.262222   0.00225133
  -1.91887   0.561202   0.0130059   0.00344561
0   0   0   0   0   0   0   0   0   3.42268   -2.17466   0.297773   -0.523944
  -1.81561   0.646124   0.0167817
0   0   0   0   0   0   0   0   0   0   3.34449   -2.19666   -0.175188
  -0.571356   -1.78564   0.659852
0   0   0   0   0   0   0   0   0   0   0   3.17096   -0.0665922   -0.209836
  -0.61294   -1.88574
0   0   0   0   0   0   0   0   0   0   0   0   3.73037   -2.15655   0.263878
  -0.00100117
0   0   0   0   0   0   0   0   0   0   0   0   0   3.20146   -2.33624   0.306799
0   0   0   0   0   0   0   0   0   0   0   0   0   0   3.06143   -2.39355
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2.84472



Q
-0.942809   -0.197911   -0.0405321   -0.00731532   -0.219874   -0.106412
  -0.0399991   -0.0133255   -0.0608525   -0.0493452   -0.0280347   -0.0140102
  -0.0196896   -0.0245177   -0.0209068   -0.0159477
0.235702   -0.904735   -0.203581   -0.0427665   0.0177029   -0.227994   -0.116967
  -0.046173   -0.0111569   -0.076226   -0.0609002   -0.0357821   -0.00965882
  -0.0331724   -0.037175   -0.0318954
0   0.254457   -0.899075   -0.210456   0.00633335   0.0254193   -0.226996
  -0.126072   -0.000622753   -0.012308   -0.07907   -0.067232   -0.00319565
  -0.0142904   -0.0385157   -0.0435329
0   0   0.256089   -0.929609   0.00183133   0.008114   0.024809   -0.246497
  0.000601282   -0.000340993   -0.0131755   -0.0833435   -0.000758023
  -0.00423422   -0.0149352   -0.0379297
0.235702   0.113092   0.0414532   0.0135052   -0.897199   -0.197654   -0.0430289
  -0.00712897   -0.232253   -0.121155   -0.0512387   -0.0202588   -0.0690997
  -0.0648985   -0.0464521   -0.0318954
0   0.254457   0.125281   0.0467055   0.284352   -0.830982   -0.200874
  -0.0452946   0.0168475   -0.243251   -0.136496   -0.0618862   -0.01575
  -0.0938816   -0.0892776   -0.068101
0   0   0.256089   0.13055   0.00579921   0.321557   -0.815828   -0.211618
  0.00806464   0.0273352   -0.242204   -0.149802   -0.00236578   -0.0197551
  -0.101953   -0.104307
```

```
0   0   0   0.26504   0.000991971   0.00703671   0.326233   -0.859918
  0.00302788   0.010944   0.0263682   -0.266142   0.000163561   -0.00264645
  -0.0212251   -0.108186
0   0   0   0   0.255814   0.146776   0.0687576   0.0301042   -0.885007
  -0.192023   -0.0404238   -0.0051388   -0.240959   -0.141195   -0.075624
  -0.0435329
0   0   0   0   0   0.285296   0.172327   0.0837416   0.302735   -0.802958
  -0.191641   -0.0417016   0.0181243   -0.2577   -0.171531   -0.104307
0   0   0   0   0   0   0.289737   0.184813   0.0130059   0.353956   -0.77962
  -0.203949   0.00931898   0.0317982   -0.258792   -0.197407
0   0   0   0   0   0   0   0.304332   0.00344561   0.0167817   0.360853
  -0.831422   0.00377804   0.0134035   0.0319874   -0.290507
0   0   0   0   0   0   0   0   0.258466   0.156023   0.0811847   0.0414052
  -0.912861   -0.24218   -0.0845132   -0.0379297
0   0   0   0   0   0   0   0   0   0.292168   0.189975   0.104167   0.319887
  -0.827524   -0.262429   -0.108186
0   0   0   0   0   0   0   0   0   0   0.299   0.20713   0.0177393   0.391245
  -0.793672   -0.290507
0   0   0   0   0   0   0   0   0   0   0   0.315362   0.00562964   0.0244623
  0.407967   -0.856434
```

Q orthornomality

q0 * q0 = 1

q0 * q1 = 9.36751e-17

q0 * q2 = 3.64292e-17

q0 * q3 = 1.12757e-17

q0 * q4 = 5.55112e-17

q0 * q5 = 5.55112e-17

q0 * q6 = 3.1225e-17

q0 * q7 = 1.60462e-17

q0 * q8 = 3.46945e-17

q0 * q9 = 3.1225e-17

q0 * q10 = 2.25514e-17

q0 * q11 = 1.9082e-17

```
q0 * q12 = 1.73472e-17

q0 * q13 = 2.60209e-17

q0 * q14 = 2.60209e-17

q0 * q15 = 2.25514e-17


q1 * q1 = 1

q1 * q2 = -1.38778e-16

q1 * q3 = -6.93889e-17

q1 * q4 = -5.55112e-17

q1 * q5 = -2.22045e-16

q1 * q6 = -1.66533e-16

q1 * q7 = -1.04083e-16

q1 * q8 = -5.37764e-17

q1 * q9 = -1.31839e-16

q1 * q10 = -1.249e-16

q1 * q11 = -9.71445e-17

q1 * q12 = -3.90313e-17

q1 * q13 = -7.97973e-17

q1 * q14 = -9.71445e-17

q1 * q15 = -1.04083e-16


q2 * q2 = 1

q2 * q3 = 1.73472e-16

q2 * q4 = 4.55365e-18

q2 * q5 = 4.16334e-17
```

```
q2 * q6 = 1.38778e-16

q2 * q7 = 1.66533e-16

q2 * q8 = 1.43115e-17

q2 * q9 = 5.37764e-17

q2 * q10 = 1.04083e-16

q2 * q11 = 1.31839e-16

q2 * q12 = 1.64799e-17

q2 * q13 = 5.63785e-17

q2 * q14 = 8.67362e-17

q2 * q15 = 1.04083e-16


q3 * q3 = 1

q3 * q4 = -6.23416e-18

q3 * q5 = 1.0842e-18

q3 * q6 = 1.38778e-17

q3 * q7 = -5.55112e-17

q3 * q8 = -1.0842e-18

q3 * q9 = 1.30104e-18

q3 * q10 = 7.80626e-18

q3 * q11 = -4.16334e-17

q3 * q12 = 2.50722e-19

q3 * q13 = 2.49366e-18

q3 * q14 = 8.67362e-19

q3 * q15 = -2.08167e-17
```

```
q4 * q4 = 1

q4 * q5 = 1.59595e-16

q4 * q6 = 8.32667e-17

q4 * q7 = 3.90313e-17

q4 * q8 = 5.55112e-17

q4 * q9 = 9.71445e-17

q4 * q10 = 8.1532e-17

q4 * q11 = 5.48606e-17

q4 * q12 = 4.16334e-17

q4 * q13 = 8.32667e-17

q4 * q14 = 9.02056e-17

q4 * q15 = 8.67362e-17


q5 * q5 = 1

q5 * q6 = 1.04083e-16

q5 * q7 = 4.85723e-17

q5 * q8 = 4.16334e-17

q5 * q9 = 5.55112e-17

q5 * q10 = 9.02056e-17

q5 * q11 = 5.0307e-17

q5 * q12 = 2.25514e-17

q5 * q13 = 5.55112e-17

q5 * q14 = 4.85723e-17
```

```
q5 * q15 = 5.89806e-17


q6 * q6 = 1

q6 * q7 = 1.31839e-16

q6 * q8 = 1.12757e-17

q6 * q9 = 4.16334e-17

q6 * q10 = 1.94289e-16

q6 * q11 = 1.59595e-16

q6 * q12 = 1.56125e-17

q6 * q13 = 5.89806e-17

q6 * q14 = 1.52656e-16

q6 * q15 = 1.8735e-16


q7 * q7 = 1

q7 * q8 = -3.03577e-18

q7 * q9 = -2.25514e-17

q7 * q10 = -6.93889e-17

q7 * q11 = -1.66533e-16

q7 * q12 = -7.80626e-18

q7 * q13 = -3.29597e-17

q7 * q14 = -8.1532e-17

q7 * q15 = -1.11022e-16


q8 * q8 = 1

q8 * q9 = -1.38778e-17
```

```
q8 * q10 = -4.16334e-17

q8 * q11 = -2.94903e-17

q8 * q12 = -5.55112e-17

q8 * q13 = -8.32667e-17

q8 * q14 = -8.32667e-17

q8 * q15 = -6.93889e-17


q9 * q9 = 1

q9 * q10 = -9.71445e-17

q9 * q11 = -6.93889e-17

q9 * q12 = -2.77556e-17

q9 * q13 = -2.77556e-17

q9 * q14 = 0

q9 * q15 = -4.85723e-17


q10 * q10 = 1

q10 * q11 = 2.498e-16

q10 * q12 = 6.07153e-18

q10 * q13 = 8.32667e-17

q10 * q14 = 2.22045e-16

q10 * q15 = 2.63678e-16


q11 * q11 = 1

q11 * q12 = 3.25261e-18

q11 * q13 = 2.77556e-17
```

```
q11 * q14 = 8.32667e-17

q11 * q15 = 1.66533e-16


q12 * q12 = 1

q12 * q13 = -1.74285e-17

q12 * q14 = -6.50521e-18

q12 * q15 = 4.33681e-18


q13 * q13 = 1

q13 * q14 = -9.71445e-17

q13 * q15 = -6.93889e-17


q14 * q14 = 1

q14 * q15 = 5.55112e-17


q15 * q15 = 1



Aqr = QR
-4   1   0   0   1   0   0   0   0   0   0   0   0   0   0   0
1   -4   1   0   0   1   0   0   0   0   0   0   0   0   0   0
0   1   -4   1   0   0   1   0   0   0   0   0   0   0   0   0
0   0   1   -4   0   0   0   1   0   0   0   0   0   0   0   0
1   0   0   0   -4   1   0   0   1   0   0   0   0   0   0   0
0   1   0   0   1   -4   1   0   0   1   0   0   0   0   0   0
0   0   1   0   0   1   -4   1   0   0   1   0   0   0   0   0
0   0   0   1   0   0   1   -4   0   0   0   1   0   0   0   0
0   0   0   0   1   0   0   0   -4   1   0   0   1   0   0   0
0   0   0   0   0   1   0   0   1   -4   1   0   0   1   0   0
0   0   0   0   0   0   1   0   0   1   -4   1   0   0   1   0
0   0   0   0   0   0   0   1   0   0   1   -4   0   0   0   1
0   0   0   0   0   0   0   0   1   0   0   0   -4   1   0   0
0   0   0   0   0   0   0   0   0   1   0   0   1   -4   1   0
0   0   0   0   0   0   0   0   0   0   1   0   0   1   -4   1
0   0   0   0   0   0   0   0   0   0   0   1   0   0   1   -4
```

```
PASSED - QR Factorization Correct
```

3 UML Diagrams

3.1 Sequence Diagram for PETSc Ex15

For this sequence diagram in Fig. 2, the ex15 has a fairly linear sequence of processes, where:

1. Declare variables, including solution vectors, the linear system matrix, the solver type, and the preconditioner type.
2. Read in user inputs and with default settings, initialize variable values.
3. Create vectors and matrix using PETSc parallelism and assemble.
4. Start linear algebra solver, calls to MatMult for initial rhs solution
5. Calls to KSPCreate, KSPSetOperators, KSPGetPC, PCSetup
6. Solve linear system with KSPSolve
7. Check error with VecNorm,KSPGetIterationNumber
8. Destroy objects (ksp,u,b)

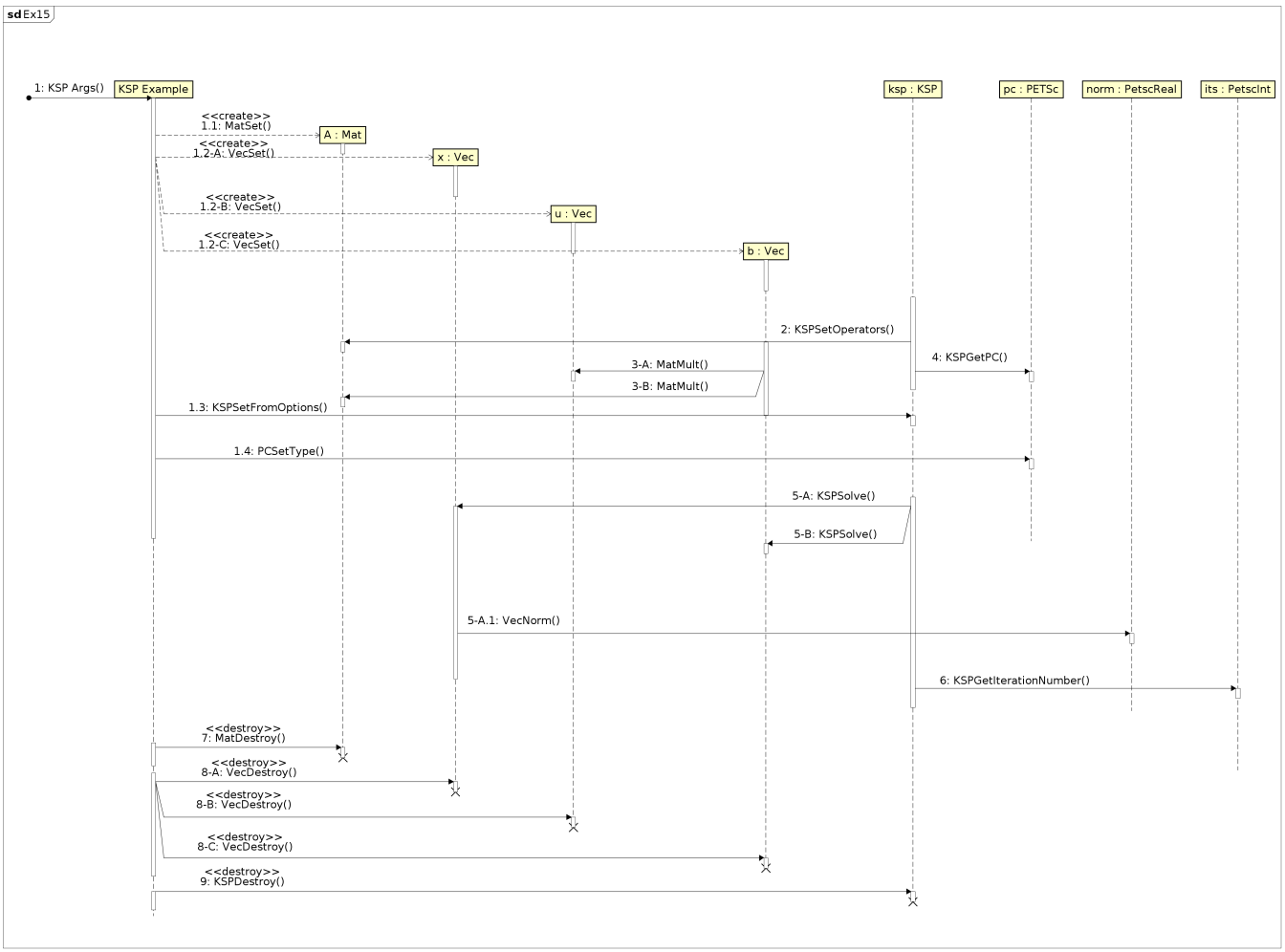

Figure 2: Sequence diagram for PETSc Ex15

3.2 Class Diagram for Abstract Linear Solver Factory

For this abstract factory class diagram in Fig. 3, the abstract products consist of the objects A,b,x, as well as the linear function solver $LinSolve()$. These products then have variants of being either in terms of dense or sparse matrices. The abstract factory interface defines the possible products to be created, with realizations for the abstract factories for the variants (Dense and Sparse). The specific products are then dependent on these variants, for example $Sparse\_A$ is dependent on the $SparseLinearSolverFactory$. An application therefore declares specific instances of these abstract products, producing realizations of the variants of the products.

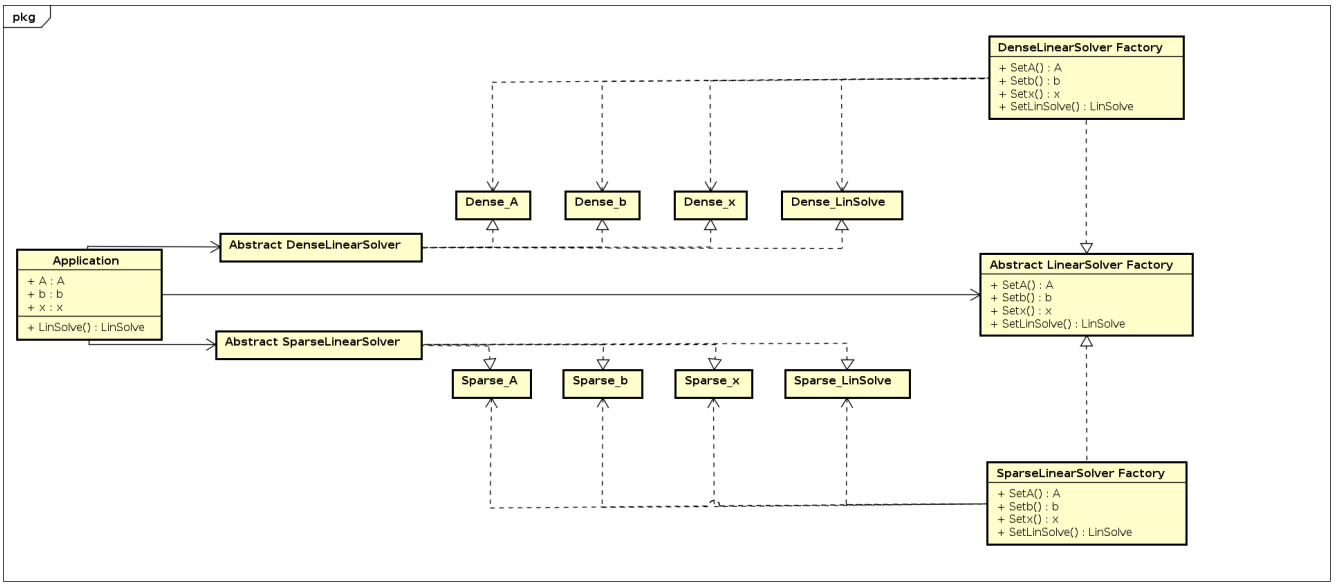

Figure 3: Abstract Factory Class diagram for linear solver.

3.3 Composite Class Diagram for Block Matrix

For this composite class diagram in Fig. 4, the relationships between a given block matrix and its sublocks is represented by calling a $GetSublock()$ operation, which returns another recursively defined block class. There are also block operations that act on the block matrix as a whole, and return another block matrix. Each block class has a shape for rows and columns. The subblock class inherits the properties of the block class by the generalization arrow, having shape attributes and block operations on itself. The recursive, composite relation of calling sublocks and returning blocks is explained by the aggregation connection between the sublock, and the multiple nested block sublocks within it. Finally, if a sublock is composed of a single element, that element is given, and has its own element operations.

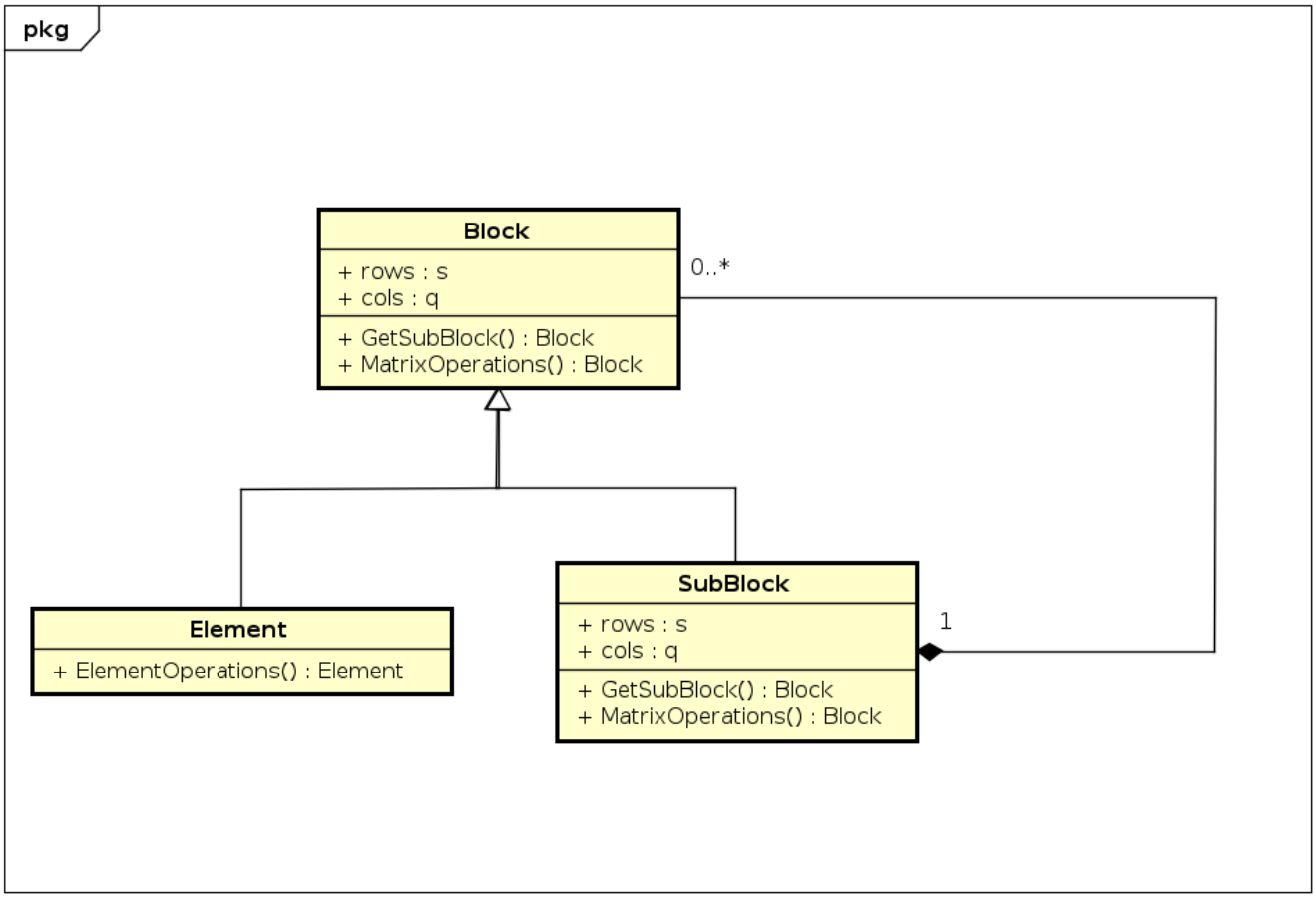

Figure 4: Composite Class diagram for block matrix.

4 Sparse Matrix - Vector Multiplication Deliverables

For the sparse matrix vector multiplication class deliverables, was assigned tasks to do with the COO sparse matrix format. Please refer to the bolded issue and pull request numbers that click-able links to the github repository.

1. Implemented all COO methods. including $MatVec()$ for matrix-vector multiplication, and $assembleStorage()$ and $unAssemble()$ for assembling the matrix in the COO format based on the abstracted $setCoefficient()$ calls. **Issue #29 and merged pull-request #64**.

2. Wrote unit tests for the $CooSparseMatrix$ class, to test for the assembly procedure and matrix-vector multiplication functioning properly. **Closed #20, and merged pull-request #65**.

3. Consulted with Eric P to fix issues in the skeleton code, and unit tests, and approved and merged the issue that Eric P implemented. **Closed Issue #17, and merged pull-request #50**.

Code 2: Fixed Iteration Methods.

```
#!/usr/bin/env python

# Python Modules
import os,sys,copy,time,pickle
import numpy as np
import scipy as sp
import matplotlib as mpl
mpl.use('agg')
from matplotlib import pyplot as plt

# Linear Solver Modules
import nelin
import pyamg
from pyamg.aggregation import smoothed_aggregation_solver


def sci(x,decimals=3,texify=True):
        if x in [None]:
                return str(x)
        s = '%.*e'%(decimals,x)
        s = s.split('e')
        s = [s[0],int(s[1].replace('-0','-').replace('+0','').replace('+',''))]
        s = r'%s%s'%(s[0],(r'\cdot10^{%d}'%s[1] if (s[1] != 0) else ''))
        if texify:
                s = r'r$%s$'%(s)
        return s

def plot(x,y,fig,ax,props):
        if x is None:
                x = np.arange(y.shape[0])
        ax.plot(x,y,**props['plot'])
        for prop in props['set']:
                getattr(ax,prop)(**props['set'][prop])
        ax.tick_params(axis='both', which='major', labelsize=30)
        return


def conditionnumber(A):
        return
          sp.sparse.linalg.norm(A)*sp.sparse.linalg.norm(sp.sparse.linalg.inv(A))

def eigenvalues(A,key,whichs=['LM']):

        n = A.shape[0]
```

```
        eigsolver = lambda A,**kwargs:
          sp.sparse.linalg.eigs(A,k=1,return_eigenvectors=False,**kwargs)

        D,L,U = DLU(A)

        if key == 'Jac':
                B = sp.sparse.linalg.spsolve(D,-(L+U))
        elif key == 'GS':
                B = sp.sparse.linalg.spsolve(D+L,-U)

        sigmas = {'LM':0.9,'SM':0.1}
        stop = False
        eigs = {which:None for which in whichs}
        for ncv in [20,50,100,1000]:
                for maxiter in [100,1000]:
                        try:
                                eigs = {which: np.abs(eigsolver(B,which=which,
                                  maxiter=maxiter*n,ncv=ncv,sigma=sigmas[which]))
                                  for which in whichs}
                                stop = True
                        except:
                                stop = False
                        if stop:
                                break
                if stop:
                        break

        if stop:
                print('Eigs Found with -- maxiter: %d, ncv: %d'%(maxiter,ncv))
        else:
                print('Eigs Not Found with -- maxiter: %d, ncv:
                  %d'%(maxiter,ncv))

        return eigs


def DLU(A):
        options = {'format':'csc'}
        D = sp.sparse.diags(A.diagonal(),**options)
        U = sp.sparse.triu(A,1,**options)
        L = sp.sparse.tril(A,-1,**options)

        return D,L,U


path = 'Ex1.pickle'
```

```
resave = False
N = [int(i) for i in [1e1,1e2,1e3,1e4]]# [1e2,1e3,1e4]]
#N = [int(i) for i in [10,20,30]]# [1e2,1e3,1e4]]
methods = ['Jac','GS']
setups = {'Jac':lambda n,m,A,x0: nelin.solutionvector(m,x0),
                  'GS':lambda n,m,A,x0: nelin.solutionvector(m,x0)
                  }

schemes = {k: getattr(nelin,k) for k in methods}
kwargs = {'Jac':{'tol':1e-9,'maxiter':1000},
                 'GS':{'tol':1e-9,'maxiter':1000,'w':1}
                 }
Nsizes = len(N)
Nmethods = len(methods)

# Data
matrix = lambda n: pyamg.gallery.poisson((n,), format='csc')
dims = lambda A: A.shape[0]
initial = lambda n: np.sin(np.arange(0,n)/100.0*np.pi*1.0) +
  np.sin(np.arange(0,n)/100.0*np.pi*6.0)
solution = lambda A,x0: np.zeros(A.shape[0])
function = lambda A,x_true: A.dot(x_true)


keys = ['x','solution','error','residual','iterations','spectral_numerical',␣
  'spectral_analytical','condition_number']


# Data
results = {}
try:
        with open(path,'rb') as fobj:
                results = pickle.load(fobj)
except:
        pass
        results = {k: {m: {n: None for n in N} for m in methods} for k in keys}

for k in keys:
        if results.get(k) is None:
                results[k] = {}
        for m in methods:
                if results[k].get(m) is None:
                        results[k][m] = {n: None for n in N}
                results[k][m].update({n:None for n in N if n not in
                  results[k][m]})

for i in range(Nsizes):
```

```
        n = N[i]

        A = matrix(n)
        m = dims(A)
        x0 = initial(n)
        x_true = solution(A,x0)
        b = function(A,x_true)

        print()
        print()

        for method in methods:

                print('Method: %s, Size: %d'%(method,n))


                _keys = ['x']
                if any([results[k][method][n] is None for k in _keys]):
                        print('Updating: ',_keys)
                        results['x'][method][n] = setups[method](n,m,A,x0)
                        schemes[method](A,b,x0=x0,
                          callback=results['x'][method][n].store,
                          **kwargs[method]);

                # Save
                if resave:
                        with open(path,'wb') as fobj:
                                pickle.dump(results,fobj)

                _keys = ['error','residual']
                if any([results[k][method][n] is None for k in _keys]):
                        print('Updating: ',_keys)
                        error,results['residual'][method][n] =
                          results['x'][method][n].getres(A,b)
                        results['error'][method][n] = error[-1]

                # Save
                if resave:
                        with open(path,'wb') as fobj:
                                pickle.dump(results,fobj)


#                 _keys = ['iterations','solution']
#                 if any([results[k][method][n] is None for k in _keys]):
#                         print('Updating: ',_keys)
```

```
#                       results['iterations'][method][n],
  results['solution'][method][n] = results['x'][method][n].getsol()

#
#                 print('Iterations: %d'%(results['iterations'][method][n]))
#
#                 # Save
#                 with open(path,'wb') as fobj:
#                         pickle.dump(results,fobj)

                _keys =
                  ['spectral_numerical','spectral_analytical','condition_number']
                if any([results[k][method][n] is None for k in _keys]):

                        results['spectral_numerical'][method][n] =
                          (results['residual'][method][n][-1]/results['residual'][method][n
                        results['spectral_analytical'][method][n] =
                          eigenvalues(A,method,['LM'])['LM']
                        results['condition_number'][method][n] =
                          conditionnumber(A)

                print('rho: %0.4e, kappa:
                  %0.4e'%(results['spectral_analytical'][method][n],
                  results['condition_number'][method][n]))

                # Save
                if resave:
                        with open(path,'wb') as fobj:
                                pickle.dump(results,fobj)

                print()
        print()

# Save
with open(path,'wb') as fobj:
        pickle.dump(results,fobj)

# Plot
options = {'fontsize':28}
plots = ['residual']
Nplots = len(plots)
# fig,axes = plt.subplots(Nmethods,Nplots)
fig,axes = plt.subplots()
fig.set_size_inches(**{'h':20,'w':20})
# axes = np.array(axes)
# axes = np.atleast_2d(axes)
# if axes.shape[0] != Nmethods:
```

```
#         axes = axes.T
_properties = {'set':{'set_xlabel':{'xlabel':'Iteration',**options},
                                            'set_yscale':{'value':'log'},
                                            'set_xscale':{'value':'log'}}}
properties = {N[0]:{'plot':{'color':'salmon','marker':'o'},
                            **_properties,
                            },
                        N[1]:{'plot':{'color':'khaki','marker':'s'},
                            **_properties,
                            },
                        N[2]:{'plot':{'color':'turquoise','marker':'^'},
                            **_properties,
                          },
                         **{n:{'plot':{'color':'violet','marker':'+'},
                            **_properties,
                          } for n in N[3:]}
                }

for i in range(Nmethods):
        method = methods[i];
        for j in range(Nplots):
                key = plots[j]
                # ax = axes[i][j]
                ax = axes
                for k in range(Nsizes):
                        n = N[k]
                        props = copy.deepcopy(properties[n])
                        if key == 'error':
                                props['set']['set_xlabel'] = {'xlabel':'Grid
                                  Index',**options}
                                props['set']['set_ylabel'] = {'ylabel':'Error
                                  (Last Iteration)',**options}
                                props['set']['set_yscale'] = {'value':'log'}
                                props['set']['set_xscale'] = {'value':'linear'}
                                props['plot']['linestyle'] = ''
                                y = np.abs(results[key][method][n][-1])

                        elif key == 'residual':
                                props['set']['set_xlabel'] =
                                  {'xlabel':'Iteration',**options}
                                props['set']['set_ylabel'] = {'ylabel':'Residual
                                  Norm',**options}
                                props['set']['set_yscale'] = {'value':'log'}
                                props['set']['set_xscale'] = {'value':'log'}
                                # props['plot']['color'] = None
                                if method =='Jac':
                                        props['plot']['linestyle'] = '--'
```

```
                                elif method == 'GS':
                                        props['plot']['color'] =
                                          'dark%s'%props['plot']['color']
                                        props['plot']['linestyle'] = '-'
                                        props['plot']['alpha'] = 0.7
                                y = results[key][method][n]

                            # props['set']['set_title'] =
                              {'label':method,**options}
                            props['plot']['label'] = r'$%s - %d ~~~ %s ~~~ %s ~~~
                              %s$'%(method,n,
                              sci(results['spectral_numerical'][method][n],3,False),




                            x = None
                            # print('Key: %s, Size: %d, Iterations:
                              %d'%(key,N[k],len(y)))
                            plot(x,y,fig,ax,props)

                            ax.legend(**{'title':r'$~~~N ~~~~~~~~ \rho_{num} ~~~~~~~~
                              \rho_{anl}~~~~~~~~ \kappa$',
                                                    'fontsize':20,
                                                    'title_fontsize':20,
                                                    'ncol':2})
fig.subplots_adjust(**{'wspace':1,'hspace':1})
fig.savefig('Ex1.pdf')
```

Code 3: QR algorithm class.

```
#include <iostream>
#include <cstddef>
#include <cmath>
#include <vector>

#include "QR.hpp"

namespace qr{

// Constructor and Destructor
template <class T>
```

```
QR<T>::QR(){
};


template <class T>
QR<T>::~QR(){
};

// Matrix Functions
template <class T>
T QR<T>::_l2norm(std::vector<T> & vec){
        T norm=0;
        for(int i=0;i<vec.size();i++){
                norm += vec[i]*vec[i];
        };
        norm = std::sqrt(norm);
        return norm;
}

 // Get column of matrix
template <class T>
void QR<T>::_column(std::vector<std::vector<T>> &arr,int j,std::vector<T> &vec){
        for(int i=0;i<vec.size();i++){
                vec[i] = arr[i][j];
        };
        return;
};

// Get row of matrix
template <class T>
void QR<T>::_row(std::vector<std::vector<T>> &arr,int i,std::vector<T> &vec){
        for(int j=0;j<vec.size();j++){
                vec[j] = arr[i][j];
        };
        return;
};

// Add vectors
template <class T>
void QR<T>::_add(std::vector<T> &vec1,std::vector<T> &vec2,T alpha,T beta){
        for(int i=0;i<vec1.size();i++){
                vec2[i] = alpha*vec1[i] + beta*vec2[i];
        };
        return;
};

// Multiply arrays
```

```
template <class T>
void QR<T>::_multiply(std::vector<std::vector<T>>
  &arr1,std::vector<std::vector<T>> &arr2,std::vector<std::vector<T>> &arr, T
  alpha){
        for (int k=0;k<arr2.size();k++){
                for(int i=0;i<arr.size();i++){
                        for(int j=0;j<arr[i].size();j++){
                                arr[i][j] += arr1[i][k]*arr2[k][j];
                        };
                };
        };
};

// Dot product between vectors
template <class T>
T QR<T>::_dot (std::vector<T> &vec1,std::vector<T> &vec2){
        T dot = 0;
        for (int i=0;i<vec1.size();i++){
                dot += vec1[i]*vec2[i];
        };
        return dot;
};

// Copy array
template <class T>
void QR<T>::_copy(std::vector<std::vector<T>> &arr1,std::vector<std::vector<T>>
  &arr2){
        for (int i=0;i<arr1.size();i++){
                for(int j=0;j<arr1[i].size();j++){
                        arr2[i][j] = arr1[i][j];
                };
        };
};

// round array
template <class T>
void QR<T>::_round(std::vector<std::vector<T>> &arr,T tol){
        for (int i=0;i<arr.size();i++){
                for(int j=0;j<arr[i].size();j++){
                        if (std::abs(arr[i][j])<tol){
                                arr[i][j] = (T)0;
                        };
                };
        };
};

// Tranpose Array
```

```
template <class T>
void QR<T>::_transpose(std::vector<std::vector<T>>
  &arr,std::vector<std::vector<T>> &arrT){
        for (int i=0;i<arr.size();i++){
                for(int j=0;j<arr[i].size();j++){
                        arrT[j][i] = arr[i][j];
                };
        };
};


//  Check if arrays are equal
template <class T>
int QR<T>::_equal(std::vector<std::vector<T>> &arr1,std::vector<std::vector<T>>
  &arr2,float tol){
        int check = 1;
        for (int i=0;i<arr1.size();i++){
                for(int j=0;j<arr1[i].size();j++){
                        if (std::abs(arr1[i][j] - arr2[i][j])>tol){
                                check = 0;
                                break;
                        };
                };
                if (check == 0){
                        break;
                };
        };

        return check;
};


template <class T>
void QR<T>::_print(std::vector<std::vector<T>> &arr){
        for(int i=0;i<arr.size();i++){
                for(int j=0;j<arr[i].size();j++){
                        std::cout << arr[i][j] << "    ";
                };
                std::cout << std::endl;
        }
};



// QR representation
template <class T>
```

```
void QR<T>::representation(std::vector<std::vector<T>>
  &A,std::vector<std::vector<T>> &Q,std::vector<std::vector<T>> &R){
        T alpha = (T)1.0;
        this->_multiply(Q,R,A,alpha);
};




// QR Factorization
template <class T>
void QR<T>::factor(std::vector<std::vector<T>> &A,std::vector<std::vector<T>>
  &Q,std::vector<std::vector<T>> &R){
        size_t m = A.size();
        size_t n = A[0].size();
        T _default = (T)0;
        T tol = (T) 1e-15;

        std::vector<std::vector<T>> V(n,std::vector<T>(m,_default));
        std::vector<std::vector<T>> QT(n,std::vector<T>(m,_default));
        std::vector<std::vector<T>> Aqr(n,std::vector<T>(n,_default));

        this->_transpose(A,V);
        this->_copy(V,QT);


        std::cout << "A" << std::endl;
        this->_print(A);
        std::cout << std::endl;

        for(int i=0;i<n;i++){
                R[i][i] = this->_l2norm(V[i]);
                this->_add(V[i],QT[i],1/R[i][i],(T)0);
                for(int j=i+1;j<n;j++){
                        R[i][j] = this->_dot(QT[i],V[j]);
                        this->_add(QT[i],V[j],-R[i][j],(T)1);
                };
        };

        this->_transpose(QT,Q);
        this->representation(Aqr,Q,R);
        this->_round(Aqr,tol);

        std::cout << "QR Factorization" << std::endl;

        std::cout << "R" << std::endl;
```

```
        this->_print(R);
        std::cout << std::endl;
        std::cout << std::endl;

        std::cout << "Q" << std::endl;
        this->_print(Q);
        std::cout << std::endl;
        std::cout << std::endl;


        std::cout << "Q orthornomality" << std::endl;
        for(int i=0;i<n;i++){
                for(int j=i;j<n;j++){
                        std::cout << "q" << i << " * " << "q" << j << " = " <<
                          this->_dot(QT[i],QT[j]) << std::endl;
                        std::cout << std::endl;
                };
                std::cout << std::endl;
        };
        std::cout << std::endl;



        std::cout << "Aqr = QR" << std::endl;
        this->_print(Aqr);
        std::cout << std::endl;
        std::cout << std::endl;


        int check = this->_equal(A,Aqr,tol);
        std::string message;
        if (check == 1){
                this->_copy(Aqr,A);
                message = "PASSED - QR Factorization Correct";
        }
        else{
                message = "ERROR - QR Factorization Incorrect";
        };
        std::cout << message << std::endl;
};



template class QR<int>;
template class QR<float>;
template class QR<double>;
```

```
};
```

Code 4: QR algorithm header.

```
#pragma once
#ifndef _QR_
#define _QR_

#include <iostream>
#include <cstddef>
#include <vector>

namespace qr{

//  Class for QR Algorithm
template <class T>
class QR {


        public:
                //  Constructor and Destructor
                QR();
                ~QR();

                // QR Factorization
                void factor(std::vector<std::vector<T>>
                  &A,std::vector<std::vector<T>> &Q,std::vector<std::vector<T>>
                  &R);
                void representation(std::vector<std::vector<T>>
                  &A,std::vector<std::vector<T>> &Q,std::vector<std::vector<T>>
                  &R);

        private:
                // Matrix functions
                T _l2norm(std::vector<T> & vec); // L2 norm
                void _column(std::vector<std::vector<T>> &arr,int
                  j,std::vector<T> &vec); // Get column of matrix
                void _row(std::vector<std::vector<T>> &arr,int i, std::vector<T>
                  &vec); // Get row of matrix
                T _dot (std::vector<T> &vec1,std::vector<T> &vec2); // Get dot
                  product of two vectors
                void _copy(std::vector<std::vector<T>>
                  &arr1,std::vector<std::vector<T>> &arr2); // Copy array
```

```
                void _multiply(std::vector<std::vector<T>>
                  &arr1,std::vector<std::vector<T>> &arr2,
                  std::vector<std::vector<T>> &arr, T alpha); // Multiply arrays
                void _add(std::vector<T> &vec1,std::vector<T> &vec2,T alpha,T
                  beta); // Add vectors
                void _transpose(std::vector<std::vector<T>>
                  &arr,std::vector<std::vector<T>> &arrT); // Tranpose array
                void _print(std::vector<std::vector<T>> &arr); // Print array
                int _equal(std::vector<std::vector<T>>
                  &arr1,std::vector<std::vector<T>> &arr2,float tol); //  Check
                  if arrays are equal
                void _round(std::vector<std::vector<T>> &arr,T tol); // Round
                  array

};
};



#endif
```

Code 5: PDE algorithm class.

```
#include <iostream>
#include <cstddef>
#include <cmath>
#include <vector>

#include "QR.hpp"



template <class T>
T randtype(){
        T r = static_cast <T> (std::rand()) / static_cast <T> (RAND_MAX);
        return r;
};


template <class T>
void printa(const std::vector<std::vector<T>> & arr){
        for(int i=0;i<arr.size();i++){
                for(int j=0;j<arr[i].size();j++){
                        std::cout << arr[i][j] << "   ";
                };
                std::cout << std::endl;
        }
```

```
        return;
};

template<class T>
void laplacian(const int dim, const size_t n,const std::string scheme){

        size_t N = n*n;
        T h = (T) 1/(N-1);
        T _zero = (T)0;

        std::vector<std::vector<T>> A(N,std::vector<T>(N,_zero));
        std::vector<std::vector<T>> Q(N,std::vector<T>(N,_zero));
        std::vector<std::vector<T>> R(N,std::vector<T>(N,_zero));

        if (scheme == "2pt"){
                const int neighbours = 2*dim;
                std::vector<int> offsets{1,-1,(int)-n,(int)n};
                size_t z = 0;
                for(int i=0;i<n;i++){
                        for(int j=0;j<n;j++){
                                z = j + i*n;
                                A[z][z] = (T) -neighbours;
                                if(i==0){
                                        if(j==0){
                                                A[z][z+1] = (T)1.0;
                                                A[z][z+n] = (T)1.0;
                                        }
                                        else if(j==(n-1)){
                                                A[z][z-1] = (T)1.0;
                                                A[z][z+n] = (T)1.0;
                                        }
                                        else{
                                                A[z][z+1] = (T)1.0;
                                                A[z][z-1] = (T)1.0;
                                                A[z][z+n] = (T)1.0;
                                        };
                                }
                                else if(i==(n-1)){
                                        if(j==0){
                                                A[z][z+1] = (T)1.0;
                                                A[z][z-n] =
                                                  (T)1.0;                    }
                                        else if(j==(n-1)){
                                                A[z][z-1] = (T)1.0;
                                                A[z][z-n] = (T)1.0;
                                        }
                                        else{
```

```
                                        A[z][z+1] = (T)1.0;
                                        A[z][z-1] = (T)1.0;
                                        A[z][z-n] = (T)1.0;
                                };
                        }
                        else{
                                if(j==0){
                                        A[z][z+1] = (T)1.0;
                                        A[z][z+n] = (T)1.0;
                                        A[z][z-n] = (T)1.0;
                                }
                                else if(j==(n-1)){
                                        A[z][z-1] = (T)1.0;
                                        A[z][z+n] = (T)1.0;
                                        A[z][z-n] = (T)1.0;
                                }
                                else{
                                        A[z][z+1] = (T)1.0;
                                        A[z][z-1] = (T)1.0;
                                        A[z][z+n] = (T)1.0;
                                        A[z][z-n] = (T)1.0;
                                };
                        };
                };
        };

}
else {
        size_t z = 0;
        for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                        z = j + n*i;
                        A[i][j] = randtype<T>();
                        std::cout << i << " " << j << " : " << A[i][j] <<
                          std::endl;
                };
        };
};

std::cout << "2pt stencil Laplacian matrix in d=2" << std::endl;
std::cout << "h^2 factor omitted for debugging" << std::endl;
std::cout << "values < 1e-15 rounded to 0 for debugging" << std::endl;
std::cout << std::endl;
qr::QR<T> qr;
qr.factor(A,Q,R);
```

```
        return;
};



int main(int argc, char *argv[]){

        size_t n = 4;
        int dim = 2;
        std::string scheme = "2pt";

        if (argc >= 2){
                n = std::atoi(argv[1]);
        };

        if (argc >= 3){
                dim = std::atoi(argv[2]);
        };
        if (argc >= 4){
                scheme = (std::string) argv[3];
        };

        laplacian<double>(dim,n,scheme);


        return 0;
};
```

Code 6: Makefile.

```
CC                  := g++
CC_STATIC          := ar
MKDIR           := mkdir -p
RMDIR           := rm -rf
ROOT           := .
FILESRC          :=
FILESTATIC          := libstatic
FILESHARED          := libshared
SRC           := $(ROOT)
OBJ           := $(ROOT)
BIN                 := $(ROOT)
INCLUDE := $(ROOT)
EXT                 := cpp
EXTDEP          := cpp
EXTOBJ          := o
EXTEXE          := out
```

```
EXTSTATIC        :=ar
EXTSHARED        :=so


SRCS     := $(wildcard $(SRC)/$(FILESRC)*.$(EXT))
DEPS          := tensor matrix QR
EXES         := pde
DEPS     := $(patsubst %,$(SRC)/%.$(EXTDEP),$(DEPS))
OBJS     := $(patsubst $(SRC)/%.$(EXT),$(OBJ)/%.$(EXTOBJ),$(SRCS))
EXES     := $(patsubst %,$(OBJ)/%.$(EXTEXE),$(EXES))

CFLAGS           :=
CLIBFLAGS         := -fPIC
STATICFLAGS                 := -rcs

# If the first argument is "run"...
ifeq (run,£(firstword £(MAKECMDGOALS)))
  # use the rest as arguments for "run"
  RUN_ARGS := $(wordlist 2,$(words $(MAKECMDGOALS)),$(MAKECMDGOALS))
  # ...and turn them into do-nothing targets
  $(eval $(RUN_ARGS)::@:)
endif


.PHONY: all run clean

$(BIN)/%.$(EXTEXE) : $(OBJ)/%.$(EXTOBJ) $(DEPS) | $(BIN)
        @echo $(DEPS)
        $(CC) -o $@ $^ $(CFLAGS)

$(OBJ)/%.$(EXTOBJ): $(SRC)/%.$(EXT) | $(OBJ)

        $(CC) -c -o $@ $^ $(CFLAGS)

$(ROOT) :
        $(MKDIR) $@

all : $(EXES)
        @echo $(OBJS)

run : $(EXES)
        ./$< $(RUN_ARGS)


# run : main.£(EXTEXE)
#         ./£< £(RUN_ARGS)
```

```
clean :
#         $(RMDIR) $(OBJS)
#         $(RMDIR) $(EXES)
```