Laboratory Assignment 3

You can use “random” or “numpy” libraries. Please do not use any other libraries. All unspecified library lines will be suppressed and if your code fails, your submission will be graded “0” points

Problem 1 – Complex numbers I [1 pts]

Write a function that takes two complex numbers and a string as inputs and returns the result of the specified operation as a complex number. You can use `numpy` library.

- Function parameter names do not matter, but the function should expect 3 parameters.
- First two parameters `(z1,z2)` are `complex`.
- Third parameter `(str)` specifies the operation, e.g. `'add'` for addition, `'sub'` for subtraction, `'mul'` for multiplication, `'div'` for division, and it should **not** be case sensitive. If the third input is not one of the mentioned operations or not given as input, then function should return `None`.
- `inputs :`
  - `z1,z2 ∈ ℂ`
  - `str ∈ { 'add'=z1+z2, 'sub'=z1-z2, 'mul'=z1*z2, 'div'=z1/z2}`
- `output : x = { x : x is a complex that is the result of the specified operation, or None}`

```
>>> z1=3+4j
>>> z2=0.5+1.7j
>>> print(problem1(z1,z2,'MUL'))
(-5.3+7.1j)
>>> print(problem1(z1,z2,'DIV'))
(2.6433121019108285-0.9872611464968153j)
>>> print(problem1(z1,z2,'SUB'))
(2.5+2.3j)
>>> print(problem1(z1,z2,'ADD'))
(3.5+5.7j)
>>> print(problem1(z1,z2,'add'))
(3.5+5.7j)
>>> print(problem1(z1,z2,'INF212'))
None
```

Problem 2 – Complex numbers II [1 pts]

Write a function that takes a complex number and a string as inputs and returns the result of the specified operation in the specified type. You can use `numpy` library.

- Function parameter names do not matter, but the function should expect 2 parameters.
- First parameter `(z)` is a `complex`.

- Second parameter (`str`) specifies the operation, e.g. `'abs'` for absolute value, `'ang'` for angle in radians, `'pol'` for polar representation, i.e. (`abs,ang`), `'con'` for conjugation, and it should **not** be case sensitive. If the second input is not one of the mentioned operations or not given as input, then function should return `None`.
- `inputs :`
  - `z ∈ ℂ`
  - `str ∈ {'abs': |z|, 'ang': ∠z, 'pol': (|z|,∠z) , 'con': z}`
- `output : x = { x : x is`
  - `a float, if str='abs',`
  - `a float, if str='deg',`
  - `a tuple, if str='pol',`
  - `a complex, if str='con',`
  - `a None, if none of the above}`

```
>>> print(problem2(z1,'ABS'))
5.0
>>> print(problem2(z1,'ANG'))
0.9272952180016122
>>> print(problem2(z1,'POL'))
(5.0, 0.9272952180016122)
>>> print(problem2(z1,'pol'))
(5.0, 0.9272952180016122)
>>> print(problem2(z1,'CON'))
(3-4j)
>>> print(problem2(z1,'INF212'))
None
```

Problem 3 – Generate a matrix with random complex elements [1 pts]

Write a function that takes two integers and two floats as inputs, and generates a numpy array with complex elements, then returns it. You can use `numpy` and `random` libraries.

- Function parameter names do not matter, but the function should expect 4 parameters.
- First two parameters (`row,column`) determine the size of the matrix, row and column, respectively.
- Third and fourth (`emin,emax`) inputs determine the interval of the real and imaginary parts of the elements.
- `inputs :`
  - `row > 1, column > 1 are integer`
  - `emin,emax are float`
- `output:`
  - `x = { x : x is a numpy array, x[i,j] ∈ [emin,emax]}`

```
>>> Zmat1=problem3(4,6,-2,2)
>>> print(Zmat1)
[[-1.93815531-1.6084903j   0.28998405+0.57240487j -0.73249031-0.23188388j
```

```
   1.17847101-1.03405855j  1.64578402+1.00460152j -0.49786926-1.31761693j]
 [ 0.87792758+1.68556854j -1.21376666+1.53855292j  0.39820565-0.90737281j
  -0.23125259-0.38171763j  0.2597682 -0.5719359j   1.31605861-1.23858856j]
 [-0.77653258-0.68162758j -1.68635892+0.93587214j  0.75185708-0.36098873j
   1.99311235-0.76316066j -0.70276077+1.77959142j  0.08318362+0.55784376j]
 [ 1.75725086+0.08863742j -0.5714548 -1.8095567j   1.46576815+0.46173301j
   0.06622199-1.41042425j -0.3108983 -1.28128319j -1.78045655-0.67284955j]]

>>> Zmat2=problem3(4,4,-2,2)
>>> print(Zmat2)
[[-1.85591837-1.37819582j -1.83785262-0.5977937j  -0.44637418-0.90824784j
   1.99867475-0.70579891j]
 [-1.38486637-0.62289749j -1.03821788-0.97841607j -1.05484428-1.55481704j
   0.25494937-1.73665365j]
 [-1.68453262+0.09977045j -0.33327246+0.30361186j  0.07496092+0.02829234j
   0.6517543 -0.20734442j]
 [-1.68764781+1.64245568j  0.06595668-0.61586396j  1.40153061-1.37552523j
  -1.76599576+0.64181031j]]
```

Problem 4 – Generate a vector with random complex elements [1 pts]

Write a function that takes an integer and two floats as inputs, and generates a numpy array with complex elements and returns it. You can use `numpy` and `random` libraries.

- Function parameter names do not matter, but the function should expect 3 parameters.
- First parameter (`row`) determines the size of the vector.
- Second and third (`emin,emax`) inputs determine the interval of the real and imaginary parts of the elements.
- `inputs :`
  - `row > 1 is integer`
  - `emin,emax are float`
- `output:`
  - `x = { x : x is a numpy array, x[i] ∈ [emin,emax]}`

```
>>> Bvec1=problem4(6,-2,2)
>>> print(Bvec1)
     [-0.45703851+0.61892251j  1.84989194+0.92653796j  1.20904843-0.65839622j
 -1.93545131-0.69088223j  1.58644867-0.63118456j  0.19009095+1.58697026j]

>>> Bvec2=problem4(4,-1,2)
>>> print(Bvec2)
[ 1.08745002-0.82469339j  0.15425774-0.32842637j  1.75631066-0.93679989j
 -0.6930785 +1.09916247j]
```

Problem 5 – Tikhonov regularization [1 pts]

Write a function that takes a numpy array and a complex/float, and returns the Tikhonov regularized matrix as a numpy array, i.e. the complex number is added to all diagonal elements of the matrix. You can use `numpy` library.

- Function parameter names do not matter, but the function should expect 2 parameters.
- First parameter (`Mat`) is a numpy array with complex elements (a matrix).
- Second parameter (`alpha`) can be float or complex.
- `inputs :`
  - `Mat = { Mat : Mat is a numpy array}`
  - `alpha is complex or float`
- `output:`
  - `x = { x : is a numpy array with the same size of Mat,`
    `diag(x)=diag(Mat)+alpha for diagonal elements,`
    `and x=Mat for off-diagonal elements}`

```
>>> problem5(Zmat2,5+2.5j)
>>> print(Zmat2)
[[ 3.14408163+1.12180418j -1.83785262-0.5977937j  -0.44637418-0.90824784j
   1.99867475-0.70579891j]
 [-1.38486637-0.62289749j  3.96178212+1.52158393j -1.05484428-1.55481704j
   0.25494937-1.73665365j]
 [-1.68453262+0.09977045j -0.33327246+0.30361186j  5.07496092+2.52829234j
   0.6517543 -0.20734442j]
 [-1.68764781+1.64245568j  0.06595668-0.61586396j  1.40153061-1.37552523j
   3.23400424+3.14181031j]]
```

Problem 6 – Solving a linear system of equations [1 pts]

Write a function that takes a numpy array as a matrix, which represents the coefficients of the equations, and a numpy array as a vector, which represents the right hand side, both with complex elements, and return the solution of the linear system of equation as a numpy array. You can use `numpy` library.

- Function parameter names do not matter, but the function should expect 2 parameters.
- First parameter (`A`) is a numpy array with complex elements (a matrix).
- Second parameter (`b`) is a numpy array with complex elements (a vector).
- If the equation system can not be solved, function should return `None`.
- `inputs :`
  - `A = { A : A is a numpy array}`
  - `b = { b : b is a numpy array}`
- `output:`
  - `x = { x : x is a numpy array and is the solution of Ax=b, or None}`

```
>>> Xvec=problem6(Zmat1,Bvec1)
>>> print(Xvec)
None
```

```
>>> Xvec=problem6(Zmat1,Bvec2)
>>> print(Xvec)
None

>>> Xvec=problem6(Zmat2,Bvec1)
>>> print(Xvec)
None

>>> Xvec=problem6(Zmat2,Bvec2)
>>> print(Xvec)
[0.60621874-0.726618j   0.44982925-0.47612599j 0.22071006-0.61249547j
 0.04311753+0.06561321j]
```

Problem 7 – Inverse of a matrix [1 pts]

Write a function that takes a numpy array as a matrix, and return the inverse of the matrix as a numpy array. You can use `numpy` library.

- Function parameter names do not matter, but the function should expect 1 parameter.
- The parameter (`A`) is a numpy array with complex elements (a matrix).
- If the inverse cannot be determined, function should return `None`.
- `input :`
  - `A = { A : A is a numpy array}`
- `output:`
  - `x = { x : x is a numpy array and is the inverse of A, or None}`

```
>>> Zinv1=problem7(Zmat1)
>>> print(Zinv1)
None

>>> Zinv2=problem7(Zmat2)
>>> print(Zinv2)
[[ 0.55402137+0.09395877j  0.21493925+0.02085295j  0.04612728+0.04166767j
  -0.12177726+0.28948677j]
 [ 0.35093807+0.09431917j  0.35673271-0.04974876j  0.07693886+0.05872883j
  -0.02940871+0.23545287j]
 [ 0.17902118-0.05861065j  0.0633704 -0.04433814j  0.17219071-0.07976463j
   0.01405615+0.12627719j]
 [ 0.10034419-0.16337231j  0.06827201-0.04948805j  0.05553671+0.06562393j
   0.22534715-0.06515784j]]
```

Problem 8 – Really inverse? [1 pts]

Write a function that takes two numpy arrays as matrices, and checks whether two matrices are inverse of each other, then returns a `boolean`. You can use `numpy` library.

- Function parameter names do not matter, but the function should expect 2 parameters.
- Both parameters (`A,B`) are numpy arrays with complex elements (matrices).

- If `A` is inverse of `B` or `B` is inverse of `A`, then return `True`, otherwise return `False`. If `A` and `B` are not comparable then return `None`.
- `inputs :`
  - `A,B = { A,B : A,B are numpy arrays}`
- `output:`
  - `x = { x : x is a boolean, or None}`

```
>>> print(problem8(Zmat2,Zinv2))
True
>>> print(problem8(Zmat2,Zmat1))
None
>>> print(problem8(Zmat2,Zmat2))
False
```

Problem 9 – Dot product `[1 pts]`

Write a function that takes a numpy array as matrix and a numpy array as vector, then returns a numpy array of multiplication of the matrix and the vector. You can use `numpy` library.

- First parameter `(A)` is a numpy array with complex elements (a matrix).
- Second parameter `(b)` is a numpy array with complex elements (a vector).
- If the multiplication cannot be performed, function should return `None`.
- `inputs :`
  - `A = { A : A is a numpy array}`
  - `b = { b : b is a numpy array}`
- `output:`
  - `x = { x : x is a numpy array and is the result of A*b, or None }`

```
>>> Bvecr=problem9(Zmat2,Xvec)
>>> print(Bvecr)
[ 1.08745002-0.82469339j  0.15425774-0.32842637j  1.75631066-0.93679989j
 -0.6930785 +1.09916247j]
>>> print(Bvec2)
[ 1.08745002-0.82469339j  0.15425774-0.32842637j  1.75631066-0.93679989j
 -0.6930785 +1.09916247j]


>>> Bvecr=problem9(Zmat2,Bvec1)
>>> print(Bvecr)
None
```

Problem 10 – Eigen do it! `[1 pts]`

Write a function that takes a numpy array as matrix and a string as inputs and returns the result of the specified operation in the specified type/order. You can use `numpy` library.

- First parameter `(A)` is a numpy array with complex elements (a matrix).

- Second parameter (`str`) specifies the operation, e.g. `'det'` for determinant, `'rank'` for rank, `'eig'` for eigenvalues, `'eigvec'` for eigenvalues and eigenvectors, `'svd'` for singular value decomposition, and it should **not** be case sensitive. If the second input is not one of the mentioned operations or not given as input, then function should return `None`.
- inputs :
  - A = { A : A is a numpy array}
  - str ∈ { 'det': determinant of A,
    'rank': rank of A,
    'eig': eigenvalues of A,
    'eigvec': eigenvalues and eigenvectors of A,
    'svd': singular value decomposition of A}
- output : x = { x : x is
  - a complex or a float, if str='det',
  - an integer, if str='rank',
  - a numpy array that holds the eigenvalues, if str='eig',
  - a tuple with length 2, that holds the eigenvalues in the first element and eigenvectors in the second element, both elements are numpy arrays, if str='eigvec',
  - a tuple with length 3, that holds the singular vectors in the first and third elements and singular values in the second element, all elements are numpy arrays, if str='svd',
  - a None, if none of the above}

```
>>> print(problem10(Zmat2,'INF212'))
None

>>> eigval=problem10(Zmat2,'EIG')
>>> print(eigval)
[1.11035872-0.2285281j  5.648586  +1.37977971j 5.23462673+3.95625085j
 3.42125746+3.20598831j]

>>> print(problem10(Zmat2,'rank'))
4

>>> Eigen=problem10(Zmat2,'EIGVEC')
>>> eigval=Eigen[0]
>>> eigvec=Eigen[1]
>>> print(eigval)
[1.11035872-0.2285281j  5.648586  +1.37977971j 5.23462673+3.95625085j
 3.42125746+3.20598831j]
>>> print(eigvec)
[[ 0.68624263+0.j         -0.2348015 -0.08758524j  0.09107927-0.5117425j
  -0.15567587-0.42385387j]
 [ 0.60612598-0.08272498j  0.83071334+0.j         -0.41748088+0.20606874j
   0.19310896+0.12165701j]
```

```
 [ 0.21423308-0.17100336j -0.20508003+0.36324549j  0.41388239-0.18827025j
  -0.22391227-0.43607188j]
 [ 0.10229428-0.26312894j  0.20988726+0.17045427j  0.55346657+0.j
   0.7097366 +0.j        ]]

>>> SVD=problem10(Zmat2,'SVD')
>>> Umat=SVD[0]
>>> Smat=SVD[1]
>>> Vmat=SVD[2]
>>> print(Umat)
[[-0.35891455-0.25914496j  0.46173578+0.10263439j -0.06525789-0.04757981j
  -0.73587386+0.17960436j]
 [-0.20150294+0.05070293j -0.70922315-0.2094849j   0.46683784-0.10728202j
  -0.42458113+0.01510748j]
 [ 0.59588583-0.12903046j  0.20906426-0.38449312j  0.16271334-0.62200583j
  -0.11820602+0.09689606j]
 [ 0.29228376-0.55108206j -0.02747465+0.19258029j  0.41262925+0.42473613j
   0.10905082+0.45878206j]]
>>> print(Smat)
[7.60198647 6.09593665 4.12677667 1.06039629]
>>> print(Vmat)
[[-0.4718184 +0.j          0.02818834-0.10193846j  0.57811744+0.40894633j
  -0.13745081+0.49583442j]
 [ 0.43499026+0.j         -0.71282243-0.06513401j  0.09186162+0.45209282j
   0.28963322+0.04136834j]
 [-0.28427756+0.j          0.32887854+0.15675518j -0.24378957+0.38872911j
   0.75419344-0.08427085j]
 [-0.71229007+0.j         -0.58524385-0.03481475j -0.22954641-0.14993806j
  -0.03307757-0.26954281j]]

>>> detA=problem10(Zmat2,'det')
>>> print(detA)
(27.10829428809462+200.9700379363007j)
```