Problem 1. *Write a multigrid V-cycle code to solve the Poisson equation in two dimensions on the unit square with Dirichlet boundary conditions. Use full weighting for restriction, bilinear interpolation for prolongation, and red-black Gauss-Seidel for smoothing. Use this code to solve*

$$\Delta u = -\exp\left(-(x-0.25)^2-(y-0.6)^2\right)$$

on the unit square $(0,1)\times(0,1)$ *with homogeneous Dirichlet boundary conditionss for different grid spacings. How many steps of pre and postsmoothing did you use? What tolerance did you use? How many cycles did it take to converge? Compare the amount of work needed to reach convergence with your solvers from Homework 3 taking into account how much work is involved in a V-cycle.*

For the given problem, I ran my multigrid V-cycle code with the stopping criterion max-norm of the residual relative to the max-norm of the RHS function, with a tolerance of 10^{-10}. Using my analysis from 2, I found that (1,1), (2,1), and (2,2) were top contenders for optimal pre-smoothing, post-smoothing step numbers, and the fastest results were obtained for 2 pre-smoothing steps and 1 post-smoothing step. The results show that iteration count for the multigrid algorithm is grid-independent. The choice of $(\nu_1,\nu_2)=(1,1)$ performs slightly better than $(2,1)$ and $(2,2)$ on most grid sizes, and is the fastest solver on the finest grid. The results are tabulated as follows:
For $(\nu_1,\nu_2)=(1,1)$,

grid spacing h	iteration count	run time (seconds)
2^{-5}	12	0.296257
2^{-6}	12	1.23618
2^{-7}	12	4.7184
2^{-8}	12	18.9813
2^{-9}	12	77.6296

For $(\nu_1,\nu_2)=(2,1)$,

grid spacing h	iteration count	run time (seconds)
2^{-5}	10	0.303806
2^{-6}	10	1.24774
2^{-7}	10	4.95247
2^{-8}	10	19.4882
2^{-9}	10	78.2237

For $(\nu_1,\nu_2)=(2,2)$,

grid spacing h	iteration count	run time (seconds)
2^{-5}	9	0.331599
2^{-6}	9	1.37659
2^{-7}	9	5.52204
2^{-8}	9	21.5597
2^{-9}	9	88.6846

Compared to the solvers from homework 3, i.e., the Jacobi, GS-lex, and SOR iterative methods, this program takes significantly fewer total iterations, and considerably less work overall. Each iteration of those methods is the same amount of work as a single smoothing operation in the multigrid V-cycle, call this amount of work W. In each V-cycle iteration of the multigrid algorithm, we do ν smoothing operations on the finest mesh. We also perform a single residual computation, restriction opeeration, and interpolation operation, each of which is comparable work to a smoothing operation, so the total work on the finest mesh is thus $(\nu+4)W$. As we discussed in class, every coarser mesh level requires a factor of 4 less work, so the total work of all the smooths in the limit of many mesh levels is

$$\sum_{\ell=1}^{L}(\nu+3)W\left(\frac{1}{4}\right)^{\ell-1} \approx (\nu+3)W\left(\frac{1}{1-\frac{1}{4}}\right) = \frac{4}{3}(\nu+3)W.$$

So for a 64×64 mesh ($h = 2^{-6}$) and tolerance of 10^{-6}, multigrid clearly outperforms the best solver from homework 3, SOR. Note that MG with (1,1) as the pre-smooth, post-smooth steps takes slightly less work than the (2,1) and (2,2) counterparts.

	iterations	work per iteration	total work
SOR	144	W	144 W
MG (1,1)	8	$\frac{4}{3}(2+4)W = 8W$	48W
MG (2,1)	6	$\frac{4}{3}(3+4)W = \frac{28}{3}W$	56W
MG (2,2)	6	$\frac{4}{3}(4+4)W = \frac{32}{3}W$	64 W

And for a 264×264 mesh ($h = 2^{-8}$) and tolerance of 10^{-7}, multigrid blows SOR out of the water. Note that MG with (1,1) pre-smooth, post-smooth steps still takes the least amount of work.

	iterations	work per iteration	total work
SOR	658	W	658 W
MG (1,1)	9	$8W$	72W
MG (2,1)	7	$\frac{28}{3}W$	65.33 W
MG (2,2)	7	$\frac{32}{3}W$	74.7 W

```
#MAT228A HW 4
#Multigrid V-Cycle Solver

#Multigrid V-cycle with
#full-weighting, bilinear interpol, GS-RB smoothing
#for Poission equation with Dirichlet BCs


from __future__ import division

import numpy as np
from math import exp, sin, pi

from GS_RB_smoother import GS_RB_smoother
from full_weighting_restriction import full_weighting_restriction
from bilinear_interpolation import bilinear_interpolation
from compute_residual import compute_residual
from direct_solve import trivial_direct_solve

def V_cycle(u, f, h, v1, v2):
  #presmooth v1 times
  u = GS_RB_smoother(u,f, h, v1)

  #compute residual
  res = compute_residual(u, f, h)

  #restrict residual
  res2 = full_weighting_restriction(res, h)

  #solve for coarse grid error, check grid level to decide whether to solve or be recursive
  if h == 2**(-2):
    error = trivial_direct_solve(res2, 2*h)
  else:
    error = np.zeros((int(1/(2*h)+1), int(1/(2*h)+1)), dtype=float)
    error = V_cycle(error, res2, 2*h, v1, v2)

  #interpolate error
  error2 = bilinear_interpolation(error, 2*h)

  #correct (add error back in)
  u = u+error2

  #post-smooth v2 times
  u = GS_RB_smoother(u, f, h, v2)

  return u
```

```
#MAT228A Homework 4
#GS-RB Smoother

#Use GS-RB to smooth
#in larger multigrid V-cycle program
#to find soluton to Poisson eqn.
#with homogeneous Dirichlet BC's

from __future__ import division

import numpy as np

def GS_RB_smoother(u, f, h, steps):

  #set number of grid points in each row/column
  n = int(1/h - 1)

  #separate red, black indices into two lists
  reds = []
  blacks = []
  for i in range(1,n+1):
    for j in range(1,n+1):
      if (i+j)%2==0:
        reds.append((i,j))
      else:
        blacks.append((i,j))

  #begin iterative scheme
  for k in range(steps):

    #loop red
    for (i,j) in reds:
      # print "red", i,j
      u[i][j] = (1/4)*(u[i-1][j]+u[i][j-1]+u[i+1][j] + u[i][j+1] - (h**2)*f[i][j])

    #loop black
    for (i,j) in blacks:
      # print "black", i,j
      u[i][j] = (1/4)*(u[i-1][j]+u[i][j-1]+u[i+1][j] + u[i][j+1] - (h**2)*f[i][j])

  return u
```

Problem 2. *Numerically estimate the average convergence factor,*

$$E_k = \left(\frac{\left\| e^{(k)} \right\|_\infty}{\left\| e^{(k)} \right\|_\infty} \right)^{1/k},$$

for different numbers of presmoothing steps, ν_1, and postsmoothing steps, ν_2, for $\nu = \nu_1 + \nu_2 \leq 4$. Be sure to use a small value of k because convergence may be reached very quickly. What test problem did you use? Do your results depend on the grid spacing? Report the results in a table, and discuss which choices of ν_1 and ν_2 give the most efficient solver.

I used my multigrid program from problem 1 to solve the problem

$$\Delta u = -2 \sin(\pi x) \sin(\pi y)$$

on the unit square $(0,1) \times (0,1)$ with homogeneous Dirichlet boundary conditions, which has the known solution

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

I performed an analysis of all the different pairings (ν_1, ν_2) for grid spacings $h = 2^{-5}, 2^{-6}, 2^{-7}$ with stopping criterion relative iterate differences with tolerance 10^{-6}, and these all achieved relatively similar results. The results do not depend on the grid spacing, as the data will testify. I report the average convergence factors for 1-5 iterations (e.g., E_3 is the average convergence factor among 3 iterations, while E_5 is the average convergence factor among 5 iterations), and 5 was chosen as the largest k to consider since my lowest reported iteration count for a multigrid solve was 8. The following table is for $h = 2^{-5}$, tolerance 10^{-10}.

(ν_1, ν_2)	E_1	E_2	E_3	E_4	iterations	run times
(0, 1)	0.305393	0.301592	0.297944	0.291537	20	0.519646
(1, 0)	0.293605	0.288987	0.291686	0.291592	24	0.68032
(1, 1)	0.11991	0.116894	0.0975584	0.156273	12	0.404654
(0, 2)	0.179352	0.176355	0.1692	0.118281	14	0.515413
(2, 0)	0.193867	0.186207	0.180232	0.149008	16	0.589619
(1, 2)	0.0805982	0.0760598	0.0647618	0.166059	10	0.390981
(2, 1)	0.0809173	0.0763635	0.064411	0.166031	10	0.405133
(3, 0)	0.140947	0.131789	0.117586	0.162013	13	0.514157
(0, 3)	0.121375	0.117422	0.0979251	0.156212	12	0.471079
(2, 2)	0.0606403	0.0543182	0.0831176	0.167628	9	0.41648
(1, 3)	0.0606286	0.0543059	0.0831232	0.167629	9	0.407873
(3, 1)	0.0607858	0.0544584	0.0830569	0.167623	9	0.443
(4, 0)	0.109314	0.100546	0.0728343	0.166327	12	0.590817
(0, 4)	0.0914259	0.0863728	0.0404893	0.164779	10	0.490951

For $h = 2^{-6}$, tolerance 10^{-10},

(ν_1, ν_2)	E_1	E_2	E_3	E_4	iterations	run times
(0, 1)	0.305759	0.303893	0.302047	0.29956	20	1.60016
(1, 0)	0.294261	0.295252	0.298011	0.300046	26	2.12789
(1, 1)	0.12071	0.119945	0.115927	0.0577091	12	1.3413
(0, 2)	0.180233	0.179461	0.177762	0.170408	14	1.42908
(2, 0)	0.194353	0.191018	0.188791	0.182702	17	1.83374
(1, 2)	0.0815659	0.0804702	0.0700874	0.111815	10	1.3723
(2, 1)	0.0818544	0.0807546	0.070458	0.11171	10	1.37951
(3, 0)	0.141378	0.137511	0.133371	0.111955	14	1.91811
(0, 3)	0.122455	0.121482	0.117529	0.0682134	12	1.613
(2, 2)	0.0616633	0.0601744	0.0327965	0.11682	9	1.46634
(1, 3)	0.0616527	0.0601636	0.0327591	0.116821	9	1.42421
(3, 1)	0.061792	0.0603017	0.0332196	0.116802	9	1.41787
(4, 0)	0.109726	0.106782	0.100253	0.113894	13	2.00789
(0, 4)	0.0926053	0.0913771	0.0837286	0.106415	10	1.51836

For $h = 2^{-7}$, tolerance 10^{-10}:

(ν_1, ν_2)	E_1	E_2	E_3	E_4	iterations
(0, 1)	0.305848	0.304463	0.303061	0.301479	21
(1, 0)	0.294425	0.297873	0.300711	0.302654	27
(1, 1)	0.120909	0.120695	0.119705	0.112979	12
(0, 2)	0.180453	0.180232	0.179786	0.178067	14
(2, 0)	0.194475	0.192975	0.192209	0.191775	18
(1, 2)	0.0818076	0.0815357	0.0792587	0.0481262	10
(2, 1)	0.0820882	0.0818156	0.0795544	0.0470683	10
(3, 0)	0.141487	0.139291	0.13751	0.133174	15
(0, 3)	0.122724	0.122481	0.121534	0.115184	12
(2, 2)	0.0619189	0.0615518	0.0572464	0.0771726	9
(1, 3)	0.0619085	0.0615412	0.057234	0.0771781	9
(3, 1)	0.0620431	0.0616766	0.0573914	0.0771083	9
(4, 0)	0.10983	0.10803	0.10601	0.095701	13
(0, 4)	0.0928997	0.0925947	0.0908535	0.0700194	10

For $h = 2^{-8}$, tolerance 10^{-10},

(ν_1, ν_2)	E_1	E_2	E_3	E_4	iterations	run times
(0, 1)	0.305871	0.304606	0.303314	0.301955	21	28.4614
(1, 0)	0.294466	0.29897	0.302007	0.304028	29	38.5927
(1, 1)	0.120959	0.120882	0.120614	0.119052	12	21.1899
(0, 2)	0.180508	0.180424	0.180285	0.179836	14	25.349
(2, 0)	0.194505	0.193764	0.193112	0.193811	19	34.2363
(1, 2)	0.0818681	0.0817999	0.081246	0.0754371	10	21.9906
(2, 1)	0.0821466	0.0820787	0.081529	0.0757927	10	22.4021
(3, 0)	0.141514	0.139864	0.139102	0.138534	16	35.8415
(0, 3)	0.122792	0.12273	0.122495	0.121037	12	26.4578
(2, 2)	0.0619828	0.0618914	0.0608837	0.038593	9	24.3085
(1, 3)	0.0619724	0.0618809	0.0608727	0.0385485	9	24.4019
(3, 1)	0.0621058	0.0620156	0.0610131	0.0391059	9	23.787
(4, 0)	0.109856	0.108447	0.107895	0.105943	14	37.0952
(0, 4)	0.0929733	0.0928971	0.0924709	0.088774	10	26.4858