Homework 5 Solution
PHZ 5156, Computational Physics
September 20, 2005

PROBLEM 1

Problem 1(a)

First check the requirements for vector addition. The obvious rule for |h>=|f>+|g> is: the vector addition of two functions produces a third function whose value at each argument x is given by h(x)=f(x)+g(x). Then vector addition turns into ordinary number addition for each value of x, and the required properties follow from the properties of ordinary number addition.

i. h(x) = f(x) + g(x) satisfies closure: h(x) is defined on [-1,1] and h(-1)=h(1).
ii. Commutivity holds: f(x)+g(x) = g(x)+f(x).
iii. Associativity holds: [f(x) +g(x)] + h(x) = f(x) + [g(x)+h(x)]
iv. The null vector is the function $f_{zero}(x)=0$ which vanishes everywhere.
v. The additive inverse of a function f(x) is $f_{inv}(x)=-f(x)$

Now the requirements for scalar multiplication. The obvious rule for |h>=α|f> is: scalar multiplication produces another function whose value at each x is given by h(x)=αf(x). Then the requirements are met by the usual commututative and associative properties of numbers.

i. h(x) = αf(x) satisfies closure: h(x) is defined on [-1,1] and h(-1)=h(1).
ii. Associativity holds: α[βf(x)] = (αβ)f(x)
iii. Distributivity in vectors holds: α[f(x)+g(x)] = αf(x) + αg(x)
iv. Distributivity in scalars holds: (α+β)f(x) = αf(x) + βf(x)
v. Identity holds: 1f(x) = f(x).

Problem 1(b)

This satisfies all the rules to be an inner product:

i. It produces a scalar.

ii. $$\langle f|g\rangle = \int_{-1}^{1} dx\ f^*(x)g(x) = \left(\int_{-1}^{1} dx\ g^*(x)f(x)\right)^* = \langle g|f\rangle^*$$

iii. Linearity:

$$\begin{aligned}\langle f|\alpha g+\beta h\rangle &= \int_{-1}^{1} dx\ f^*(x)\left[\alpha g(x)+\beta h(x)\right]\\ &= \alpha\int_{-1}^{1} dx\ f^*(x)g(x) + \beta\int_{-1}^{1} dx\ f^*(x)h(x)\\ &= \alpha\langle f|g\rangle + \beta\langle f|h\rangle\end{aligned}$$

Problem 1(c)

Use $\langle e_n|e_m\rangle = \int_{-1}^{1} dx\ e_n^*(x)e_m(x) = \frac{1}{2}\int_{-1}^{1} dx\ e^{i(-n+m)\pi x}$.

If m=n this gives 1. If m is not equal to n, this produces zero because $e^{ik\pi}=(-1)^k$ if k is an integer.

Problem 1(d)
My code for this homework is in one big file with the following structure:

```
#Homework 5
#Computational Physics PHZ 5156
#September 2005
from scipy import *
from scipy.integrate import odeint
import time
import Gnuplot,Gnuplot.funcutils
from LinearAlgebra import *

def problem1d():
        ....
def problem1e():
        ....
#Main program
problem1d()
#problem1e() ....
```

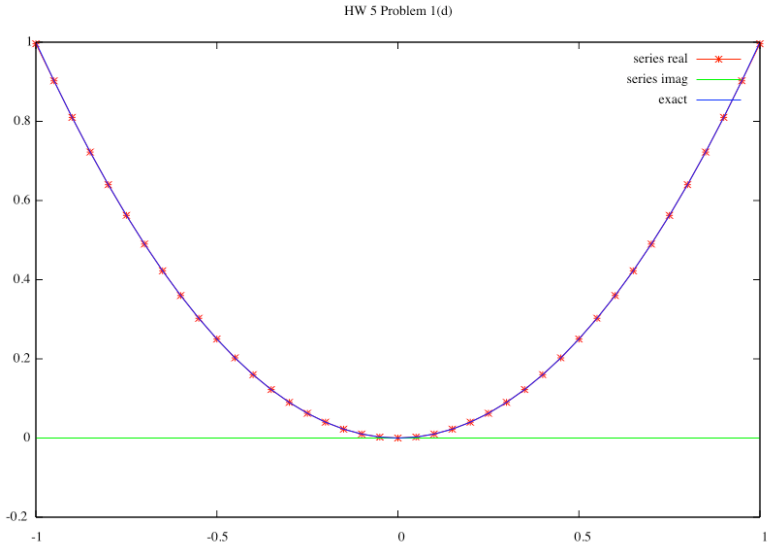
The calculation for each problem is in a separate function, and the main body simply calls each function.

In the following I will only show the function that solves each problem.

Code for problem 1(d)

```
def problem1d():
        I = 0.+1.j
        h = 0.05
        x = arange(-1.,1.+h,h)
        fapprox = 0.*x
        nmax = 100
        for n in arange(-nmax,nmax+1):
                if n==0:
                        fn = sqrt(2.)/3.
                else:
                        fn = sqrt(8.) * ((-1)**n) / ( (n*pi)**2 )
                fapprox = fapprox + fn*exp(I*n*pi*x)/sqrt(2.)
        fapproxreal = real(fapprox)
        fapproximag = imag(fapprox)
        f = x**2
        g = Gnuplot.Gnuplot(debug=0)
        g.title('HW 5 Problem 2')
        g('set data style lines')
        g1 = Gnuplot.Data(x,f,title='exact')
        g2 = Gnuplot.Data(x,fapproxreal,title='series real',with='linespoints 1 3')
        g3 = Gnuplot.Data(x,fapproximag,title='series imag')
        g.plot(g2,g3,g1)
        return
```

The plot below makes it obvious that the function being expanded here is x^2, so this code plots x^2 as well as the expansion.



Problem 1(e)

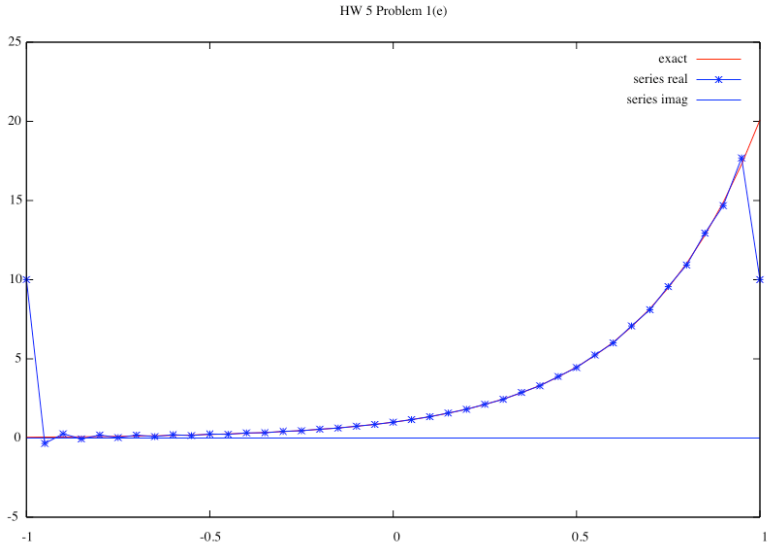
Here you need to solve an integral to get f_n=<e_n|f> for each n:

$$f_n = \langle e_n | f \rangle = \int_{-1}^{1} dx e_n^*(x) f(x) = \frac{1}{\sqrt{2}} \int_{-1}^{1} dx e^{-in\pi x} e^{3x} = (-1)^n \frac{\sqrt{2}}{3 - in\pi} \sinh 3$$

Code for Problem 1(e)

```
def problem1e():
        I = 0.+1.j
        h = 0.05
        x = arange(-1.,1.+h,h)
        f = exp(3.*x)
        nmax = 100
        fapprox = 0.*x
        for n in arange(-nmax,nmax+1):
                fn = ((-1)**n) * sqrt(2.) * sinh(3.)/(3.-I*n*pi)
                fapprox = fapprox + fn*exp(I*n*pi*x)/sqrt(2.)
        fapproxreal = real(fapprox)
        fapproximag = imag(fapprox)
        g = Gnuplot.Gnuplot(debug=0)
        g.title('HW 5 Problem 1(e)')
        g('set data style lines')
        g1 = Gnuplot.Data(x,f,title='exact')
        g2 = Gnuplot.Data(x,fapproxreal,title='series real',with='linespoints 3 3')
        g3 = Gnuplot.Data(x,fapproximag,title='series imag')
        g.plot(g1,g2,g3)
        return
```

The plot shows good agreement if nmax is big enough:



Notice that the imaginary part is essentially zero. The real part of the expansion agrees well with the exact value, except at the endpoints x=0,x=1. This happens because the basis functions $e_n(x)$ used here are equal at x=0,1 while this is not true for $f(x)=e^{3x}$. That is, f(x) is not in the vector space! But at every point besides x=0,1 the expansion can be as close to the exact value as you want by keeping enough terms in the sum (i.e., making nmax big enough).

PROBLEM 2

Problem 2(a)
This works exactly like Problem 1(a). The key question is closure; it is easy to see that f(x)+g(x) and αf(x) vanish at x=0 and x=1, hence closure is satisfied.

Problem 2(b)
This works exactly like Problem 1(b). The limits on the integrals change, which has no effect.

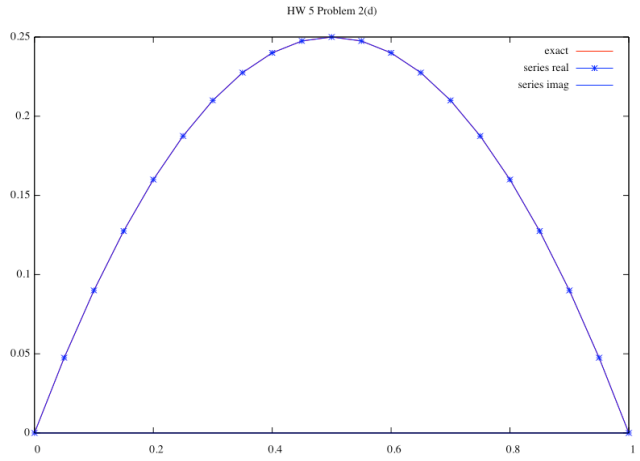
Problem 2(c)
This requires doing integrals of the form $\langle e_n|e_m\rangle$ for n,m=1,2,3,…. It is easy to see using trig identities (or better yet by looking the integrals up in a table) that $\langle e_n|e_m\rangle=\delta_{nm}$.

Problem 2(d)
The coefficients are found by evaluating $f_n=\langle e_n|f\rangle$, which for each n is a definite integral:

$$f_n = \langle e_n | f \rangle = \int_0^1 dx e_n^*(x) f(x) = \sqrt{2}\int_0^1 dx \sin(n\pi x) x(1-x)$$
$$= \begin{cases} \dfrac{4\sqrt{2}}{(n\pi)^3}, & n = \text{odd} \\ 0, & n = \text{even} \end{cases}$$

Code for Problem 2(d)

```
def problem2d():
        I = 0.+1.j
        h = 0.05
        x = arange(0.,1.+h,h)
        f = x*(1-x)
        nmax = 100
        fapprox = 0.*x
        for n in arange(1,nmax+1,2):
                fn = sqrt(32.) / ((n*pi)**3)
                fapprox = fapprox + fn*sqrt(2.)*sin(n*pi*x)
        fapproxreal = real(fapprox)
        fapproximag = imag(fapprox)
        g = Gnuplot.Gnuplot(debug=0)
        g.title('HW 5 Problem 2(d)')
        g('set data style lines')
        g1 = Gnuplot.Data(x,f,title='exact')
        #g('set data style points')
        g2 = Gnuplot.Data(x,fapproxreal,title='series real',with='linespoints 3 3')
        g3 = Gnuplot.Data(x,fapproximag,title='series imag')
        g.plot(g1,g2,g3)
        return
```



Problem 2(e)

The idea here is to sandwich the operator between the vectors (i.e., functions) and watch what happens:

$$\Omega_{mn} = \left\langle e_m \middle| \hat{\Omega} e_n \right\rangle = 2\int_0^1 dx\, \sin(m\pi x)\frac{d^2}{dx^2}\sin(n\pi x)$$

$$= 2\int_0^1 dx\, \sin(m\pi x)(n\pi)^2 \sin(n\pi x)$$

$$= (n\pi)^2 2\int_0^1 dx\, \sin(m\pi x)\sin(n\pi x) = (n\pi)^2 \langle e_m | e_n \rangle = (n\pi)^2 \delta_{mn}$$

Hence the upper-left corner of the infinite-dimensional matrix Ω is

$$\Omega = \pi^2 \begin{pmatrix} 1 & 0 & 0 & \cdots \\ 0 & 4 & 0 & \cdots \\ 0 & 0 & 9 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

PROBLEM 3

This set of functions with the obvious rule for vector addition is not a vector space. If h(x)=f(x)+g(x) then h(0)=h(1)=6. Thus h(x) is not in the original set of functions, and so closure is not satisfied. Showing that one requirement is not met is sufficient to prove this is not a vector space.

PROBLEM 4

The key point of this is to understand that symbols such as <i| mean the Hermitian conjugate of symbols such as |i>. Then each step is simply matrix multiplication.

(a) It is easy to see that <1|1>=<2|2>=<3|3>=1 while <1|2>=<1|3>=<2|3>=0; the former means the vectors are normalized and the latter that they are mutually orthogonal; hence the set is orthonormal: $\langle i|j\rangle = \delta_{ij}$. Here is an example:

$$\langle 1|2\rangle = \frac{1}{\sqrt{6}}\frac{1}{\sqrt{5}}\begin{pmatrix} 1 & -i & 2 \end{pmatrix}\begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{30}}(2+0-2) = 0$$

(b) Using the first three inner products written above, this sum is <1|1>+<2|2>+<3|3> which equals 3.

(c) This means multiplying column |1> times row <1| plus similar terms for states 2 and 3. The result is the identity matrix, something that happens for any orthonormal basis:

$$\sum_{i=1}^{3}|i\rangle\langle i| = \frac{1}{6}\begin{pmatrix} 1 \\ i \\ 2 \end{pmatrix}\begin{pmatrix} 1 & -i & 2 \end{pmatrix} + \frac{1}{5}\begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix}\begin{pmatrix} 2 & 0 & -1 \end{pmatrix} + \frac{1}{30}\begin{pmatrix} -1 \\ 5i \\ -2 \end{pmatrix}\begin{pmatrix} -1 & -5i & -2 \end{pmatrix}$$

$$= \frac{1}{6}\begin{pmatrix} 1 & -i & 2 \\ i & 1 & 2i \\ 2 & -2i & 4 \end{pmatrix} + \frac{1}{5}\begin{pmatrix} 4 & 0 & -2 \\ 0 & 0 & 0 \\ -2 & 0 & 1 \end{pmatrix} + \frac{1}{30}\begin{pmatrix} 1 & 5i & 2 \\ -5i & 25 & -10i \\ 2 & 10i & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

PROBLEM 5

Problem 5 Code

```
def problem5():
        A = array( ((1,5,0,0),(2,3,4,5),(7,6,5,4),(1,5,9,2)) , Float)
        print "Matrix A=\n",A
        print "Det(A)=",determinant(A)
        B = inverse(A)
        print "Inv(A)=\n",B
        AB = matrixmultiply(A,B)
        print "AB=\n",AB
        print "AB (rounded) =\n",round(AB)
        return
```

Output

```
Matrix A=
[[ 1.  5.  0.  0.]
 [ 2.  3.  4.  5.]
 [ 7.  6.  5.  4.]
 [ 1.  5.  9.  2.]]

Det(A)= -891.0

Inv(A)=
[[-0.11111111 -0.14590348  0.20763187 -0.05050505]
 [ 0.22222222  0.0291807  -0.04152637  0.01010101]
 [-0.11111111 -0.0650954   0.01571268  0.13131313]
 [ 0.          0.29292929 -0.07070707 -0.09090909]]

AB=
[[  1.00000000e+00   0.00000000e+00   5.55111512e-17  -6.93889390e-18]
 [  0.00000000e+00   1.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   1.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   2.77555756e-17   1.00000000e+00]]

AB (rounded) =
[[ 1.  0.  0. -0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

PROBLEM 6

Problem 6 code

```
def problem6():
        H = array( ((5,3,1,0.),(3,7,2,1),(1,2,1,3),(0,1,3,9)) , Float)
        print "H=\n",H,"\n"

        print
        print "Problem 6(a)"
        evalues, evectors = eigenvectors(H)
        print "evalues=",evalues,"\n"
        print "evectors=\n",evectors,"\n"

        print "Problem 6(b)"
        for i in arange(len(evalues)):
                print "Check evector number ",i,":\n"
                print "           H*evec =",dot(H,evectors[i])
                print "   evalue*evec =",evalues[i]*evectors[i]
                print "   difference    =",dot(H,evectors[i])-evalues[i]*evectors[i]
                print

        print "Problem 6(c)"
        print "Should get eigenvalues on the diagonal and zeros elsewhere."
        evectors = transpose(evectors)  # python gave a transpose to begin with
        ans = matrixmultiply(H,evectors)
        evectors_hermitian = conjugate(transpose(evectors))
        ans = matrixmultiply(evectors_hermitian,ans)
        print "Vdagger * H * V =\n", ans
        print "\n","Rounded"
        print round(ans,4)
        return
```

Output

```
H=
[[ 5.  3.  1.  0.]
 [ 3.  7.  2.  1.]
 [ 1.  2.  1.  3.]
 [ 0.  1.  3.  9.]]

Problem 6(a)
evalues= [ 11.25662313   8.33053552   2.78434844  -0.37150709]

evectors=
[[ 0.31678335  0.54697734  0.34106204  0.69580226]
 [ 0.49894799  0.5685507  -0.0438881  -0.65259087]
 [ 0.80365377 -0.58489358 -0.02593598  0.10661819]
 [-0.06956888 -0.18832257  0.93865747 -0.28038712]]

Problem 6(b)
Check evector number  0 :
        H*evec = [ 3.56591084  6.15711779  3.83920684  7.83238376]
  evalue*evec = [ 3.56591084  6.15711779  3.83920684  7.83238376]
  difference    = [  5.32907052e-15  -2.66453526e-15   2.66453526e-15  -8.88178420e-16]

Check evector number  1 :
        H*evec = [ 4.15650393  4.73633177 -0.36561134 -5.43643145]
  evalue*evec = [ 4.15650393  4.73633177 -0.36561134 -5.43643145]
  difference    = [ -3.55271368e-15  -4.44089210e-15  -2.77555756e-16   5.32907052e-15]

Check evector number  2 :
        H*evec = [ 2.23765213 -1.62854754 -0.0722148   0.29686219]
  evalue*evec = [ 2.23765213 -1.62854754 -0.0722148   0.29686219]
  difference    = [  4.44089210e-16   2.44249065e-15   1.67921232e-15   9.99200722e-16]

Check evector number  3 :
        H*evec = [ 0.02584533  0.06996317 -0.3487179   0.1041658 ]
  evalue*evec = [ 0.02584533  0.06996317 -0.3487179   0.1041658 ]
  difference    = [ -8.32667268e-17  -7.07767178e-16  -5.55111512e-17  -1.52655666e-16]

Problem 6(c)
Should get eigenvalues on the diagonal and zeros elsewhere.
Vdagger * H * V =
[[  1.12566231e+01  -2.66453526e-15   1.99840144e-15  -4.16333634e-16]
 [ -1.77635684e-15   8.33053552e+00   1.27675648e-15  -3.74700271e-16]
 [  2.77555756e-15   9.99200722e-16   2.78434844e+00   3.81639165e-16]
 [ -8.88178420e-16  -2.22044605e-16   4.16333634e-16  -3.71507088e-01]]

Rounded
[[ 11.2566  -0.       0.      -0.    ]
 [ -0.       8.3305   0.      -0.    ]
 [  0.       0.       2.7843   0.    ]
 [ -0.      -0.       0.      -0.3715]]
```

PROBLEM 7

Problem 7 code

```
def problem7():
        H0 = array( ((5,2,0,0),(2,4,0,0),(0,0,1,1),(0,0,1,2)) , Float)
        print "H0=\n",H0
        V = 0.01* array( ((1,2,1,1),(2,3,0,2),(1,0,3,1),(1,2,1,2)) )
        print "V=\n",V
        evalues0 = eigenvalues(H0)
        evalues = eigenvalues(H0+V)
        evalues0 = sort(evalues0)
        evalues = sort(evalues)
        print "evalues0=",evalues0
        print "evalues =",evalues
        print "difference = ", evalues - evalues0
        return
```

Output

```
H0=
[[ 5.  2.  0.  0.]
 [ 2.  4.  0.  0.]
 [ 0.  0.  1.  1.]
 [ 0.  0.  1.  2.]]
V=
[[ 0.01  0.02  0.01  0.01]
 [ 0.02  0.03  0.    0.02]
 [ 0.01  0.    0.03  0.01]
 [ 0.01  0.02  0.01  0.02]]
evalues0= [ 0.38196601  2.43844719  2.61803399  6.56155281]
evalues = [ 0.40016772  2.44135896  2.64977435  6.59869897]
difference =  [ 0.01820171  0.00291177  0.03174036  0.03714616]
```

Can see that the difference between the eigenvalues is of order the magnitude of elements of V.

PROBLEM 8

Problem 8 code

```
def problem8():
        A = array( ((1,3,0,0),(3,4,0,0),(0,0,5,3),(0,0,3,8)) , Float)
        A1 = array( ((1,3),(3,4)), Float)
        A2 = array( ((5,3),(3,8)), Float)
        print "A=\n",A
        evalues,evectors = eigenvectors(A)
        print "evalues=\n",evalues
        print "evectors=\n",evectors
        print "A1=\n",A1
        evalues,evectors = eigenvectors(A1)
        print "evalues=\n",evalues
        print "evectors=\n",evectors
        print "A2=\n",A2
        evalues,evectors = eigenvectors(A2)
        print "evalues=\n",evalues
        print "evectors=\n",evectors
        return
```

Output

```
A=
[[ 1.  3.  0.  0.]
 [ 3.  4.  0.  0.]
 [ 0.  0.  5.  3.]
 [ 0.  0.  3.  8.]]
evalues=
[-0.85410197  5.85410197  3.14589803  9.85410197]
evectors=
[[-0.85065081  0.52573111 -0.         -0.        ]
 [-0.52573111 -0.85065081  0.          0.        ]
 [ 0.          0.         -0.85065081  0.52573111]
 [-0.         -0.         -0.52573111 -0.85065081]]
A1=
[[ 1.  3.]
 [ 3.  4.]]
evalues=
[-0.85410197  5.85410197]
evectors=
[[-0.85065081  0.52573111]
 [-0.52573111 -0.85065081]]
A2=
[[ 5.  3.]
 [ 3.  8.]]
evalues=
[ 3.14589803  9.85410197]
evectors=
[[-0.85065081  0.52573111]
 [-0.52573111 -0.85065081]]
```

The answers all agree. You could solve the 2x2 problems by hand. Then the actual eigenvectors are the 2D eigenvectors padded appropriately with zeros. For instance the first eigenvector from A1 corresponds to this eigenvector for A:

[-0.85065081 0.52573111 **0 0**]