1 Question 1

1.1 Explanation

The program uses simpsons integration and bisection root finding to evaluate the limits of the integral. By creating an equation where the intgral minus a half is equal to zero we use the bisection rule to find which limits bring that integral value close to a half. We keep doing this until we have attained the five digits in precision of a and I.

1.2 Code:

```
double gaussian(double x);
double gaussian_integral(double a);
double linear(double x);

int main(int argc, char **argv){
double precision_a = 1.0;
double precision_I = 1.0;
double dp = 1;
double step_size =1;
double previous_I = 0;
    double previous_a = 0;

printf("|%7s|\t |%7s|\t |%7s|\t|%7s|\n"
        ,"a","integral","precision of a","precision of I");
while (precision_a > (1/pow(10,5)) || precision_I > (1/pow(10,5))){
double a = bisection(gaussian_integral,0 , PI, dp);
        double I = simp(-a,a,gaussian,step_size);
        precision_I = fabs(I - previous_I);
        precision_a = fabs(a -previous_a);
        dp+=1;
        step_size = 5*step_size;
        previous_I = I;
        previous_a = a;
        printf("%.7f\t %.7f\t %.7f\t \t%.7f\n", a,I,precision_a,precision_I);
}
}

double gaussian(double x){
  return (1/sqrt(2.0*PI))*1/(exp(pow(x,2)/2));
}

double linear(double x){
return cos(x);
}

double gaussian_integral(double a){
     return simp(-a,a,gaussian,100) - .5;
}
```

1.3 Output:

```
|      a|  |integral|  |precision of a| |precision of I|
```

```
0.6745201  0.2857897  0.6745201   0.2857897
0.6745201  0.4688317  0.0000000   0.1830419
0.6745201  0.4941995  0.0000000   0.0253678
0.6745201  0.4988720  0.0000000   0.0046725
0.6745201  0.4997905  0.0000000   0.0009185
0.6745201  0.4999736  0.0000000   0.0001831
0.6745201  0.5000102  0.0000000   0.0000366
0.6745201  0.5000175  0.0000000   0.0000073
```

From this output we determine $a = 0.6745201$ and that is accurate to five digits of precision due to the fact that both the values in a and I after the last iteration changed by an order less than five digits.

2 Question 2

2.1 Explanation

For the given anhamornic oscilitor, this program integrates over the ode with both a fourth order sympletic algorighm and fourth order runge-kutta. At each iteration the program outputs the deviation of the energy produce from each algorighm from the orginal energy. At the same time keeps track of the maximum deviation.

2.2 Code:

```
double force(double x);
void derivatives(double* x, double* y);

int main(int argc, char **argv){
    FILE *f = fopen("output.txt","w");

    //intialize PERFL variables
    double* q = calloc(2, sizeof(double));
    q[0] = 1.0;
    q[1] = 0.0;
    //intialize rk4 variables
    ODE rk4 = new_ODE(2);
    setq(rk4,0,1.0);
    setq(rk4,1,0.0);
    set_func(rk4, derivatives);

    double h = .02*PERIOD;
    double step_limit;
    double energy_rk,energy_lf,max_dev_lf,max_dev_rk ;
    energy_rk = energy_lf =  max_dev_lf = max_dev_rk = 0.0;

    printf("How many periods: ");
    scanf("%lf",&step_limit);
    step_limit = step_limit*50;

    fprintf(f,"t/T \t E_L \t E_K  \n");
    for (int i = 0;i<=step_limit;i++){
        runge_kutta_4_update(rk4, h);
        energy_rk = pow(getq(rk4,0),4)/4.0 + pow(getq(rk4,1),2)/2.0;
        position_verlet(q,force,h);
```

```
        energy_lf= pow(q[0],4)/4.0 + pow(q[1],2)/2.0;
        double dev_lf =fabs(energy_lf-0.25);
        double dev_rk =fabs(energy_rk-0.25);
        if(dev_rk > max_dev_rk){max_dev_rk=dev_rk;}
        if(dev_lf > max_dev_lf){max_dev_lf=dev_lf;}
        fprintf(f,"%f \t %.12f \t %.12f \n",(i*h)/PERIOD,dev_lf,dev_rk);

    }
    printf("Maximum deviation for PEFRL = %f\n",max_dev_lf);
    printf("Maximum deviation for runge-kutta = %f\n",max_dev_rk) ;
    return 0;

}
void derivatives(double* y, double* x){
    x[0] = y[1]; //x dot
    x[1] = -pow(y[0],3); // p dot
}


double force(double x){
    return -1.0*pow(x,3);
}
```

2.3 Output:

```
How many periods: 1
t     E_L   E_K
0.000000   0.000006185886   0.000000318386
0.148327   0.000021853204   0.000000158207
0.296653   0.000040525033   0.000000044327
0.444980   0.000056456447   0.000000381800
...............
6.823027   0.000040717489   0.000004989798
6.971354   0.000022049311   0.000005825687
7.119681   0.000006313113   0.000007040121
7.268007   0.000000000738   0.000008002920
7.416334   0.000006059780   0.000008321670


Maximum deviation for PEFRL = 0.000078
Maximum deviation for runge-kutta = 0.000008

How many periods: 1000

t     E_L   E_K
0.000000   0.000006185886   0.000000318386
0.148327   0.000021853204   0.000000158207
0.296653   0.000040525033   0.000000044327
0.444980   0.000056456447   0.000000381800
0.593307   0.000067196799   0.000001187569
.............
7415.740833   0.000078062801   0.007696753017
```

```
7415.889160   0.000078064487   0.007696940377
7416.037487   0.000078064446   0.007696758081
7416.185813   0.000078051298   0.007696700748
7416.334140   0.000077974911   0.007697089620


Maximum deviation for PEFRL = 0.000078
Maximum deviation for runge-kutta = 0.007697
```

It goes to show that at low periods the runge kutta algorighm maybe more precision since its energy deviates, but at higher time periods RK4 deviation growths while PERFL seems to ocsiliate around the intial energy.

3 Question 3

3.1 Explanation

3.2 Code:

```
double force_y(double y,double r);
double force_x(double x,double r);
double energy(double r, double v_y, double v_x);
double angular_momentum(double r, double theta, double v_x, double v_y);

int main(int argc, char **argv){
    double* q_x = calloc(2, sizeof(double));
    double* q_y = calloc(2, sizeof(double));
    printf("Intial position in x : ");
    scanf("%lf",&q_x[0]);
    printf("Intial velocity in y : ");
    scanf("%lf",&q_y[1]);
    FILE *f = fopen("output.txt","w");
    FILE *f_1 = fopen("out3.txt","w");
    //q_x[0] = 1.0; // position
    q_x[1] = 0.0; // momentum
    q_y[0] = 0.0;
    //q_y[1] = 1.0;

    int n = 10000;
    double h = (2*PI)/n;
    double int_E = energy(sqrt(pow(q_x[0],2) + pow(q_y[0],2)),q_x[1],q_y[1]);
    double int_L = angular_momentum(q_x[0],q_y[0],q_x[1],q_y[1]);
    fprintf(f,"x \t y \t R  \n");
    fprintf(f_1,"%s \t %s \t %s \t %s \t %s\n","Time","E","L","Deviation in E","Deviation in L");
    for (int i = 0;i<=n;i++){
        double radius = sqrt(pow(q_x[0],2) + pow(q_y[0],2));
        double theta = atan2(q_y[0],q_x[0]);
        double E = energy(radius,q_x[1],q_y[1]);
        double L = angular_momentum(q_x[0],q_y[0],q_x[1],q_y[1]);
        fprintf(f,"%f \t %f \t %f\n",q_x[0] ,q_y[0],radius);
        fprintf(f_1,"%.5e \t %.5e \t %.5e\t",i*h,E,L);
        fprintf(f_1,"%.6e \t %.16e\n",fabs(E-int_E), fabs(L-int_L));
```

```
        PERLF(q_x,force_x,h,radius);
        PERLF(q_y,force_y,h,radius);


    }
    fclose(f);
    return 0;

}

double energy(double r, double v_y, double v_x){
    return (pow(v_x,2)+pow(v_y,2))/2.0 - (1.0/r);
}
double angular_momentum(double x, double y, double v_x, double v_y){
    return x*v_y - y*v_x;
}

double force_y(double y,double r){
    return (-1*y)/(pow(r,3));
}

double force_x(double x,double r){
    return (-1*x)/(pow(r,3));
}
```

3.3 Output:

```
Intial position in x : 1
Intial velocity in y : .7

Time   E   L   Deviation in E   Deviation in L
0.00000e+00   -7.55000e-01   7.00000e-01 0.000000e+00   0.0000000000000000e+00
6.28319e-04   -7.55000e-01   7.00000e-01 1.521006e-14   0.0000000000000000e+00
1.25664e-03   -7.55000e-01   7.00000e-01 1.522116e-13   0.0000000000000000e+00
1.88496e-03   -7.55000e-01   7.00000e-01 5.321299e-13   1.1102230246251565e-16
2.51327e-03   -7.55000e-01   7.00000e-01 1.276979e-12   0.0000000000000000e+00
3.14159e-03   -7.55000e-01   7.00000e-01 2.508105e-12   1.1102230246251565e-16
3.76991e-03   -7.55000e-01   7.00000e-01 4.347522e-12   1.1102230246251565e-16
4.39823e-03   -7.55000e-01   7.00000e-01 6.916578e-12   2.2204460492503131e-16
5.02655e-03   -7.55000e-01   7.00000e-01 1.033706e-11   1.1102230246251565e-16
.................

6.28067e+00   -7.64000e-01   7.00000e-01 8.999898e-03   2.6645352591003757e-15
6.28130e+00   -7.64000e-01   7.00000e-01 8.999935e-03   2.5535129566378600e-15
6.28193e+00   -7.64000e-01   7.00000e-01 8.999973e-03   2.4424906541753444e-15
6.28256e+00   -7.64000e-01   7.00000e-01 9.000010e-03   2.4424906541753444e-15
6.28319e+00   -7.64000e-01   7.00000e-01 9.000048e-03   2.4424906541753444e-15



Intial position in x : 1
Intial velocity in y : 1
=
Time   E   L    Deviation in E    Deviation in L
```

```
0.00000e+00   -5.00000e-01   1.00000e+00 0.000000e+00   0.0000000000000000e+00
6.28319e-04   -5.00000e-01   1.00000e+00 0.000000e+00   0.0000000000000000e+00
1.25664e-03   -5.00000e-01   1.00000e+00 2.220446e-16   1.1102230246251565e-16
1.88496e-03   -5.00000e-01   1.00000e+00 1.110223e-16   1.1102230246251565e-16
2.51327e-03   -5.00000e-01   1.00000e+00 1.110223e-16   1.1102230246251565e-16
3.14159e-03   -5.00000e-01   1.00000e+00 4.440892e-16   3.3306690738754696e-16
3.76991e-03   -5.00000e-01   1.00000e+00 4.440892e-16   2.2204460492503131e-16
4.39823e-03   -5.00000e-01   1.00000e+00 2.220446e-16   1.1102230246251565e-16
5.02655e-03   -5.00000e-01   1.00000e+00 2.220446e-16   2.2204460492503131e-16
.........

6.28067e+00   -5.00000e-01   1.00000e+00 4.773959e-15   4.8849813083506888e-15
6.28130e+00   -5.00000e-01   1.00000e+00 4.551914e-15   4.6629367034256575e-15
6.28193e+00   -5.00000e-01   1.00000e+00 4.551914e-15   4.6629367034256575e-15
6.28256e+00   -5.00000e-01   1.00000e+00 4.662937e-15   4.8849813083506888e-15
6.28319e+00   -5.00000e-01   1.00000e+00 4.662937e-15   4.6629367034256575e-15
```

4 Question 4

4.1 Explanation

This program uses an Rk4 method, decoupling the 2nd order schroedinger equation into the first derivative of the wave amplitude and the wave amplitude we step foward from 0 to 1. We do this for varying step sizes.

4.2 Code:

```
double potential(double x);
void differential_func(double* parameters, double* result,double x);

int main(int argc, char **argv){
double intial_amplitude= 0.0;
double intial_amp_velocity =1.0;
double h[4] = {0.2,.1,.05,.025};
    ODE wave_eq = new_ODE(2);
    set_func(wave_eq,differential_func);
    for(int k = 0; k<4;k++){
       printf("Step size equals %f\n",h[k] );
       printf("------------------------------------------------------\n");
       double position = 0.0;
       setq(wave_eq,0,intial_amplitude);
       setq(wave_eq,1,intial_amp_velocity);
       for (int i = 0; i <= (int)(1.0/h[k]); ++i)
       {
     printf("x = %f \t psi = %f psi_prime = %f \n",position,getq(wave_eq,0),getq(wave_eq,1) );
     position+= h[k];
     runge_kutta_4_update(wave_eq,h[k],position);
       }
       printf("\n \n");
    }
return 0;
}

void differential_func(double* parameters, double* result,double x){
```

```
result[0] = parameters[1];
result[1] = -(pow(PI,2))*parameters[0];
}

double potential(double x){
if(x>0 && x<1){
return 0;
}else{
return INFINITY;
}
}
```

4.3 Output:

```
Step size equals 0.200000
--------------------------------------------------------
x = 0.000000   psi = 0.000000 psi_prime = 1.000000
x = 0.200000   psi = 0.186841 psi_prime = 0.809102
x = 0.400000   psi = 0.302346 psi_prime = 0.310104
x = 0.600000   psi = 0.302569 psi_prime = -0.306633
x = 0.800000   psi = 0.187517 psi_prime = -0.806047
x = 1.000000   psi = 0.001118 psi_prime = -0.997964


Step size equals 0.100000
--------------------------------------------------------
x = 0.000000   psi = 0.000000 psi_prime = 1.000000
x = 0.100000   psi = 0.098355 psi_prime = 0.951058
x = 0.200000   psi = 0.187083 psi_prime = 0.809035
......
x = 0.800000   psi = 0.187139 psi_prime = -0.808859
x = 0.900000   psi = 0.098424 psi_prime = -0.950932
x = 1.000000   psi = 0.000078 psi_prime = -0.999934


Step size equals 0.050000
--------------------------------------------------------
x = 0.000000   psi = 0.000000 psi_prime = 1.000000
x = 0.050000   psi = 0.049794 psi_prime = 0.987688
x = 0.100000   psi = 0.098363 psi_prime = 0.951057
......
x = 0.900000   psi = 0.098367 psi_prime = -0.951050
x = 0.950000   psi = 0.049799 psi_prime = -0.987684
x = 1.000000   psi = 0.000005 psi_prime = -0.999998


Step size equals 0.025000
--------------------------------------------------------
x = 0.000000   psi = 0.000000 psi_prime = 1.000000
x = 0.025000   psi = 0.024974 psi_prime = 0.996917
x = 0.050000   psi = 0.049795 psi_prime = 0.987688
.......
x = 0.950000   psi = 0.049795 psi_prime = -0.987688
```

```
x = 0.975000   psi = 0.024975 psi_prime = -0.996917
x = 1.000000   psi = 0.000000 psi_prime = -1.000000
```

We can see that as we make the step size smaller the value of $\psi(1)$ is approaching zero. Which shows that smaller step sizes make the Rk4 algorighm more accurate. This is due to the fact that the algorighm advances by evaulating derivates at that step size.

5 Question 5

5.1 Explanation

This program will create an an array of random numbers and then call heapsort.c . Heapsort first builds a heap by starting at $\frac{N}{2}$ and checking the heap property in this case that element i is larger that the element at either 2i or 2i +1(known as children). By doing this we have the largest element at the top of the heap. We then enter the second phase where we swap the last element with the top element and then check if it larger than the children if not we swap those elements recursively. After the elements are sorted we run a simple bisection algorighm (or binary search) where we start from the end and begining look at the middle and depending if the element we are looking for is less or more than than he middle we recusively search into that section.

5.2 Code:

```
void bisection(double value, double* heap,int hi,int lo);
int main(int argc, char **argv){
int order;
printf("How many random numbers: 10^");
scanf("%d",&order);
int size =(int)pow(10,order);
double* heap = calloc(size,sizeof(double*));
srand(time(NULL));
for(int i = 1; i<size ;i++){
    double R =(rand()/(RAND_MAX+1.0));
    heap[i] = R;
    //printf("%f\n",R);
}

    heapsort(heap,size);
for(int i = 1; i<6 ;i++){
printf("%f \n",heap[i] );
}
printf(".........\n");
for(int i = size-7; i<size-1 ;i++){
printf("%f \n",heap[i] );
}
    bisection(.7,heap,size,0);
    printf("Number of operations: ");
    num_ops();
return 0;
}

void bisection(double value, double* heap,int hi,int lo){
```

```
int middle = (hi+lo)/2;
if (middle == hi || middle == lo){
printf("Found it between elements %d %d\n These have values %f %f\n",lo,hi, heap[lo],heap[hi]);
return;
}
if(heap[middle]<value){
//printf("Go into %d to %d\n", middle,hi );
bisection(value,heap,hi,middle);
}else{
//printf("Go into %d to %d\n", lo,middle );
bisection(value,heap,middle,lo);
}

}
```

5.3 Output:

```
How many random numbers: 10^6
0.000001
0.000002
0.000004
0.000005
0.000006
.........
0.999998
0.999999
0.999999
0.999999
0.999999
0.999999
Found it between elements 700496 700497
 These have values 0.699999 0.700000
Number of operations: 19549150



How many random numbers: 10^7
0.000000
0.000000
0.000000
0.000000
0.000000
.........
0.999999
1.000000
1.000000
1.000000
1.000000
1.000000
Found it between elements 6997915 6997916
 These have values 0.700000 0.700000
Number of operations: 228834702
```

When running heap sort on 10^6 elements versus 10^7 there is only a multiplitive factor of 11.7 between the number of operation needed to be done. This is measured by the amount of times we must check the parent

to the children.

6 Question 6

6.1 Explanation

This program reads in x,y, and error values from a file. By calculating the values for the matrix U represented by an array. After that we a_0 the intercept and a_1 the slope and their respective errors are simply computed from this array. Finally the function chi sq() calculates the χ^2 value. Afterwards for part c the program calculate the Quality by computing $1 - \frac{1}{\Gamma(\frac{M}{2})} \int_0^{\frac{\chi^2}{2}} y^{\frac{M}{2}-1} e^{-y} dy$

6.2 Code:

```
void linear_fit(int N,double* y, double* x, double* err);
double chi_calc(int N,double a_0,double a_1,double* x,double* y, double* err);
double quality_calc(int dof, double chi_sq,double (*half_gamma)(double));
double half_gamma(double y);

int main(int argc, char **argv){
FILE* f = fopen("data.txt","r");
double temp[600];
double x[200];
double y[200];
double err[200];
int count =0.0;

while(fscanf(f,"%lf",&temp[count])!=EOF){
       count +=1;
}

    for(int i =0;i<600;i++){
     int colum_num = i%3;
     if(colum_num==0){x[i/3]=temp[i];}
        if(colum_num==1){y[i/3]=temp[i];}
     if(colum_num==2){err[i/3]=temp[i];}
    }
for(int i =0;i<200;i++){
//printf("%lf\t%lf\t%lf\n",x[i],y[i],err[i] );

    }
    linear_fit(200,y,x,err);
fclose(f);

}


void linear_fit(int N,double* y, double* x, double* err){
double U[3];//may need to change for more general
double v[2];

for(int i =0;i<=3;i++){ // 00 01=01 11
for(int j = 0;j<N;j++){
```

```
U[i] += pow(x[j],i);///pow(err[j],2); //x_i ^alpha+beta
     }
}

for(int i = 0;i<2;i++){
for(int j = 0;j<N;j++){
v[i] += y[j]*pow(x[j],i);///pow(err[j],2);
    }
}

double delta = U[0]*U[2] - pow(U[1],2);
double a_0 = (U[2]*v[0] - U[1]*v[1])/delta;
double a_1 = (-U[1]*v[0] + U[0]*v[1])/delta;
double sigma_a_0 = U[0]/delta;
double sigma_a_1 = U[2]/delta;

printf("a_0 = %f +/- %f , a_1=%f +/- %f\n",a_0,sigma_a_0,a_1,sigma_a_1);
printf("Chi Square = %f\n", chi_calc(N,a_0,a_1,x,y,err));
printf("There are N=%d total variables and two constraints a_0 and a_1\n",N);
    printf("therefore there are N-2=%d degrees of freedom ",N-2);
    printf("so chi_squared/dof =%f\n",chi_calc(N,a_0,a_1,x,y,err)/(N-2));
    printf("Quality = %f\n", quality_calc(N-2,chi_calc(N,a_0,a_1,x,y,err),half_gamma));

}

double chi_calc(int N,double a_0,double a_1,double* x,double* y, double* err){
double chi_sq;
for (int i = 0; i < N; ++i)
{
chi_sq += pow((y[i]-a_0-a_1*x[i])/err[i],2);
}
return chi_sq;

}

double quality_calc(int dof, double chi_sq,double (*half_gamma)(double)){
       return 1.0- (1.0/tgamma((double)(dof/2.0)))*simp(0.0, chi_sq/2.0, half_gamma,10);

}
double half_gamma(double y){
       return pow(y,(198/2.0-1))*exp(-y);
}
```

6.3 Output:

```
a_0 = 0.904589 +/- 0.000002 , a_1=5.002239 +/- 0.020151
Chi Square = 205.917073
There are N=200 total variables and two constraints a_0 and a_1
therefore there are N-2=198 degrees of freedom so chi_squared/dof =1.039985
Quality = 0.325519
```

We take to be the number of degree of freedom to be the number of x variables minus the two constraints from a_0, a_1. We find the quality factor to be .32 which proves it to be a good fit.

6.4 PERLF.c

```
#define THETA 1.35120719195966

void position_verlet(double* q,double (*force)(double), double h ){
q[0] = q[0] + THETA*(h/2.0)*q[1];
q[1] = q[1] + THETA*h*(force)(q[0]);
    //printf("test %.15f\n",(force)(q[0]));
//printf("xps = %.12f pps = %.12f \n theta=%f\n",q[0],q[1],THETA);
    q[0] = q[0] + (1.0-THETA)*(h/2.0)*q[1];
    q[1] = q[1] + (1.0-2.0*THETA)*h*(force)(q[0]);
    q[0] = q[0] + (1-THETA)*(h/2.0)*q[1];
    q[1] = q[1] + THETA*h*(force)(q[0]);
    q[0] = q[0] + THETA*(h/2.0)*q[1];

}
```