1. Consider the nonlinear Schrödinger equation

$$\begin{cases} iv_t = v_{xx} + 2|v|^2 v & \text{for } (x,t) \in \mathbf{R} \times (0,T) \\ v(x,0) = f(x) & \text{for } x \in \mathbf{R}. \end{cases}$$

Suppose for some $L > 0$ that $f(x) = f(x+L)$ for $x \in \mathbf{R}$. Prove the resulting solution v satisfies $v(x,t) = v(x+L,t)$ for all $(x,t) \in \mathbf{R} \times (0,T)$. We shall say that v satisfies the nonlinear Schrödinger equation with L-periodic boundary conditions.

Consider the function $w(x,t) = v(x+L,t)$. Since f is L-periodic we have

$$w(x,0) = v(x+L,0) = f(x+L) = f(x).$$

By the chain rule

$$w_{xx}(x,t) = \frac{\partial^2}{\partial x^2} v(x+L,t) = v_{xx}(x+L,t).$$

Therefore $w(x,t)$ satisfies the differential equation

$$\begin{cases} iw_t = w_{xx} + 2|w|^2 w & \text{for } (x,t) \in \mathbf{R} \times (0,T) \\ w(x,0) = f(x) & \text{for } x \in \mathbf{R}. \end{cases}$$

Uniquencess of solutions implies that $w(x,t) = v(x,t)$. Therefore $v(x,t) = v(x+L,t)$ for all $(x,t) \in \mathbf{R} \times (0,T)$ which was to be shown.

2. Let v be a solution to the nonlinear Schrödinger equation with L-periodic boundary conditions. Let $\Delta x = L/K$ and $\Delta t = T/N$ and consider the finite difference method for approximating v on $[0, L] \times [0, T]$ given by

$$\begin{cases} u_k^{n+1} = u_k^{n-1} - \dfrac{2i\Delta t}{\Delta x^2}\delta^2 u_k^n - 4i\Delta t|u_k^n|^2 u_k^n & \text{for } n = 1, \ldots, N-1 \\ & \text{and } k = 1, \ldots, K \\ u_k^1 = u_k^0 - \dfrac{i\Delta t}{\Delta x^2}\delta^2 u_k^0 - 2i\Delta t|u_k^0|^2 u_k^0 & \text{for } k = 1, \ldots, K \\ u_0^n = u_K^n \quad \text{and} \quad u_{K+1}^n = u_1^n & \text{for } n = 0, \ldots, N \\ u_k^0 = f(k\Delta x) & \text{for } k = 1, \ldots, K. \end{cases}$$

For $L = 10$ and $T = 0.2$ choose K and N sufficiently large to approximate $v(5, 0.2)$ to at least 3 significant digits for the personalized initial condition f given below

$$\begin{aligned} f_{\text{Alexander}}(x) &= 2\exp\big(3i\sin(\omega x)\big) + \sin(3\omega x) \\ f_{\text{Anthony}}(x) &= \exp(i\omega x) - \sin(2\omega x) + \exp(5i\omega x) \\ f_{\text{Brian}}(x) &= 0.5\exp(i\omega x) + 0.5\sin(2\omega x) - 2i\cos(3\omega x) \\ f_{\text{Jordan}}(x) &= 2\exp(-i\omega x) - i\cos(3\omega x) \\ f_{\text{Joseph}}(x) &= 0.5\exp(i\omega x) - \exp(2i\omega x) + i\exp(3i\omega x) + 0.5\exp(5i\omega x) \\ f_{\text{Kyle}}(x) &= i\sin(\omega x) + \cos(2\omega x) + \exp(-3i\omega x) \\ f_{\text{Masakazu}}(x) &= 1 + \exp(i\omega x) \\ f_{\text{Sarah}}(x) &= \exp(i\omega x) + \exp(-2i\omega x) + \exp(3i\omega x) \\ f_{\text{Shijie}}(x) &= \sin\big(3\cos(\omega x)\big) + 2\exp(2i\omega x) \end{aligned}$$

where $\omega = \pi/5$.

To implement these scheme the driver and common routines were factored into the file `utility.c` with header `utility.h` while the file named `classical.c` was used for the method itself. The driver and common routines are

```
1  /*  utility.c -- Utility functions for Programming Project 2
2      Written May 2013 by Eric Olson  */
3
4  #include <stdio.h>
5  #include <math.h>
6  #include <complex.h>
7  #include "utility.h"
8
9  double dx,dt;
10
11 #define FD(func,def) static Complex func(double x){ \
12     if(x<0) printf("%s=%s\n",#func,#def); \
13     return def; }
```

```
14
15 FD(fAlexander,2*cexp(3*I*sin(W*x))+sin(3*W*x))
16 FD(fAnthony,cexp(I*W*x)-sin(2*W*x)+cexp(5*I*W*x))
17 FD(fBrian,0.5*cexp(I*W*x)+0.5*sin(2*W*x)-2*I*cos(3*W*x))
18 FD(fJordan,2*cexp(-I*W*x)-I*cos(3*W*x))
19 FD(fJoseph,
20 0.5*cexp(I*W*x)-cexp(2*I*W*x)+I*cexp(3*I*W*x)+0.5*cexp(5*I*W*x))
21 FD(fKyle,I*sin(W*x)+cos(2*W*x)+cexp(-3*I*W*x))
22 FD(fMasakazu,1+cexp(I*W*x))
23 FD(fSarah,cexp(I*W*x)+cexp(-2*I*W*x)+cexp(3*I*W*x))
24 FD(fShijie,sin(3*cos(W*x))+2*cexp(2*I*W*x))
25
26 static double energy(int K,Complex u[K]){
27     int k;
28     double energy=0.0;
29     for(k=0;k<K;k++){
30         energy+=creal(u[k]*conj(u[k]));
31     }
32     return energy/K;
33 }
34
35 static Complex init(int K,Complex u[K+2],Complex (*f)(double)){
36     int k;
37     for(k=0;k<K;k++) {
38         double xk=k*dx;
39         u[k]=f(xk);
40     }
41     u[K]=u[0]; u[K+1]=u[1];
42 }
43
44 typedef struct {
45     Complex (*f)(double);
46     char *p;
47 } param;
48 #define FN(x) { .f=x,.p=#x }
49 void everyone(int K,Complex scheme(int,Complex *,int),int N){
50     dx=L/K;
51     dt=T/N;
52     Complex u[K+2];
53     char bufR[128],bufI[128];
54     param P[]= {FN(fAlexander),FN(fAnthony),FN(fBrian),FN(fJordan),
55         FN(fJoseph),FN(fKyle),FN(fMasakazu),FN(fSarah),FN(fShijie)};
56     int J=sizeof(P)/sizeof(param);
57     int j;
58     for(j=0;j<J;j++) P[j].f(-1);
```

```
59      printf("\nL=%g K=%d dx=%g\nT=%g N=%d dt=%g\n",L,K,dx,T,N,dt);
60      sprintf(bufR,"Re(u(%g,%g))",K/2*dx,T);
61      sprintf(bufI,"Im(u(%g,%g))",K/2*dx,T);
62      printf("\n%-12s %14s %14s %14s %14s\n",
63          "Name","|u(.,0)|","|u(.,T)|",bufR,bufI);
64      for(j=0;j<J;j++){
65          init(K,u,P[j].f);
66          double e1=energy(K,u);
67          Complex r=scheme(K,u,N);
68          double e2=energy(K,u);
69          printf("%-12s %14.6e %14.6e %14.6e %14.6e\n",
70              P[j].p,e1,e2,creal(r),cimag(r));
71      }
72  }
```

with header

```
1  /*  utility.h -- Utility functions for Programming Project 2
2      Written May 2013 by Eric Olson  */
3
4  #ifndef _UTILITY_H
5  #define _UTILITY_H
6
7  #include <complex.h>
8
9  typedef double complex Complex;
10
11 #define T 0.2
12 #define L 10.0
13 #define W (2*M_PI/L)
14
15 extern double dx,dt;
16 extern void everyone(int K,Complex scheme(int,Complex *,int),int N);
17
18 #endif
```

and the classical explicit method was coded as

```
1  /*  classical.c -- Programming Project 2
2      Nonlinear Schroedinger using Classical Explicit Method
3
4      This is method 1(a)(i) described in
5
6      Thiab Taha, Mark Ablowitz, Analytical and Numerical Aspects of
7      Certain Nonlinear Evolution Equations II.  Numerical, Nonlinear
8      Schrodinger Equation, J. of Comp. Physics, 55 (1984), 203-230.
9  */
```

```
10
11 #include <stdio.h>
12 #include <math.h>
13 #include "utility.h"
14
15 Complex classical(int K,Complex u[K+2],int N){
16     Complex w[K+2],unm1[K+2];
17     int k;
18     double rho=dt/dx/dx;
19     int n;
20     for(k=1;k<=K;k++){
21         w[k]=-I*(rho*(u[k+1]-2*u[k]+u[k-1])
22             +2*dt*(u[k]*conj(u[k]))*u[k]);
23     }
24     w[0]=w[K]; w[K+1]=w[1];
25     for(k=0;k<=K+1;k++){
26         unm1[k]=u[k]+w[k];
27     }
28     for(n=1;n<N;n++){
29         for(k=1;k<=K;k++){
30             w[k]=-I*(2*rho*(u[k+1]-2*u[k]+u[k-1])
31                 +4*dt*(u[k]*conj(u[k]))*u[k]);
32         }
33         w[0]=w[K]; w[K+1]=w[1];
34         for(k=0;k<=K+1;k++){
35             Complex tmp=u[k];
36             u[k]=unm1[k]+w[k];
37             unm1[k]=tmp;
38         }
39     }
40     return u[K/2];
41 }
42
43 main(){
44     printf("classical -- Programming Project 2\n"
45         "Nonlinear Schroedinger using Classical Explicit Method\n\n");
46     everyone(800,classical,400000);
47 }
```

The output is

```
classical -- Programming Project 2
Nonlinear Schroedinger using Classical Explicit Method

fAlexander=2*cexp(3*I*sin(W*x))+sin(3*W*x)
fAnthony=cexp(I*W*x)-sin(2*W*x)+cexp(5*I*W*x)
fBrian=0.5*cexp(I*W*x)+0.5*sin(2*W*x)-2*I*cos(3*W*x)
```

```
fJordan=2*cexp(-I*W*x)-I*cos(3*W*x)
fJoseph=0.5*cexp(I*W*x)-cexp(2*I*W*x)+I*cexp(3*I*W*x)+0.5*cexp(5*I*W*x)
fKyle=I*sin(W*x)+cos(2*W*x)+cexp(-3*I*W*x)
fMasakazu=1+cexp(I*W*x)
fSarah=cexp(I*W*x)+cexp(-2*I*W*x)+cexp(3*I*W*x)
fShijie=sin(3*cos(W*x))+2*cexp(2*I*W*x)

L=10 K=800 dx=0.0125
T=0.2 N=400000 dt=5e-07

Name               |u(.,0)|       |u(.,T)|   Re(u(5,0.2))   Im(u(5,0.2))
fAlexander     4.500000e+00   4.499996e+00  -1.440933e+00  -2.922288e+00
fAnthony       2.500000e+00   2.500006e+00  -1.310246e+00   4.137590e-01
fBrian         2.375000e+00   2.374995e+00   1.851442e+00   1.965779e+00
fJordan        4.500000e+00   4.499997e+00   1.875292e+00   4.649261e-01
fJoseph        2.500000e+00   2.499999e+00  -8.733876e-01  -4.058580e-01
fKyle          2.000000e+00   2.000001e+00   1.263949e-01  -3.491421e-01
fMasakazu      2.000000e+00   1.999996e+00   2.377104e-03  -7.211769e-02
fSarah         3.000000e+00   3.000011e+00  -5.794036e-01   2.397391e-01
fShijie        4.424677e+00   4.424663e+00   1.267973e+00  -1.485885e+00
```

Note that values for N and K were chosen as a trade off between accuracy, stability and computational time. The output is consistent with the output from the more efficient methods used in the next secions.

3. Approximate $v(5, 0.2)$ using one of the other methods found, for example, in

> Thiab R. Taha, Mark J. Ablowitz, Analytical and Numerical Aspects of Certain Nonlinear Evolution Equations II: Numerical, Nonlinear Schrödinger Equation, *J. Comput. Phys.*, **55** (1984), pp. 203–230.

Please include a bibliographic reference to the method you implemented as a comment in your source code.

I implemented the hopscotch method 1(a)(ii) and the split step Fourier method 2(i) following Taha and Ablowitz. The code for the hopscotch method is

```
/*  hopscotch.c -- Programming Project 2
    Nonlinear Schroedinger using the Hopscotch Method

    This is method 1(a)(ii) described in

    Thiab Taha, Mark Ablowitz, Analytical and Numerical Aspects of
    Certain Nonlinear Evolution Equations II.  Numerical, Nonlinear
    Schrodinger Equation, J. of Comp. Physics, 55 (1984), 203-230.
*/

#include <stdio.h>
#include <math.h>
#include "utility.h"

Complex hopscotch(int K,Complex u[K+2],int N){
    Complex w[K+2],q[K+2];
    int k;
    double rho=dt/dx/dx;
    int n;
    for(n=1;n<=N;n++){
        if(n==1){
            for(k=1;k<=K;k+=2){
                w[k]=u[k]-I*rho*(u[k+1]-2*u[k]+u[k-1])
                    -I*dt*(u[k-1]*conj(u[k-1])*u[k-1]
                          +u[k+1]*conj(u[k+1])*u[k+1]);
            }
            w[K+1]=w[1];
        } else {
            for(k=n%2;k<=K+1;k+=2){
                w[k]=2*u[k]-q[k];
            }
        }
        for(k=n%2+1;k<=K;k+=2){
            w[k]=1/(1-2*I*rho)*(u[k]-I*rho*(w[k-1]+w[k+1])
                -I*dt*(w[k-1]*conj(w[k-1])*w[k-1]
```

```
                        +w[k+1]*conj(w[k+1])*w[k+1]));
        }
        if(n%2==1) w[K+1]=w[1];
        else w[0]=w[K];
        for(k=0;k<=K+1;k++) {
            q[k]=u[k];
            u[k]=w[k];
        }
    }
    return u[K/2];
}

main(){
    printf("hopscotch -- Programming Project 2\n"
        "Nonlinear Schroedinger using the Hopscotch Method\n\n");
    everyone(800,hopscotch,400000);
}
```

with output

```
hopscotch -- Programming Project 2
Nonlinear Schroedinger using the Hopscotch Method

fAlexander=2*cexp(3*I*sin(W*x))+sin(3*W*x)
fAnthony=cexp(I*W*x)-sin(2*W*x)+cexp(5*I*W*x)
fBrian=0.5*cexp(I*W*x)+0.5*sin(2*W*x)-2*I*cos(3*W*x)
fJordan=2*cexp(-I*W*x)-I*cos(3*W*x)
fJoseph=0.5*cexp(I*W*x)-cexp(2*I*W*x)+I*cexp(3*I*W*x)+0.5*cexp(5*I*W*x)
fKyle=I*sin(W*x)+cos(2*W*x)+cexp(-3*I*W*x)
fMasakazu=1+cexp(I*W*x)
fSarah=cexp(I*W*x)+cexp(-2*I*W*x)+cexp(3*I*W*x)
fShijie=sin(3*cos(W*x))+2*cexp(2*I*W*x)

L=10 K=800 dx=0.0125
T=0.2 N=400000 dt=5e-07

Name              |u(.,0)|       |u(.,T)|   Re(u(5,0.2))   Im(u(5,0.2))
fAlexander     4.500000e+00   4.497342e+00  -1.427328e+00  -2.931356e+00
fAnthony       2.500000e+00   2.498831e+00  -1.309640e+00   4.121534e-01
fBrian         2.375000e+00   2.374884e+00   1.846055e+00   1.966600e+00
fJordan        4.500000e+00   4.495430e+00   1.887079e+00   4.708170e-01
fJoseph        2.500000e+00   2.499696e+00  -8.730900e-01  -4.060822e-01
fKyle          2.000000e+00   1.999463e+00   1.264794e-01  -3.493688e-01
fMasakazu      2.000000e+00   2.000082e+00   2.485139e-03  -7.206665e-02
fSarah         3.000000e+00   3.000680e+00  -5.800543e-01   2.412936e-01
fShijie        4.424677e+00   4.422633e+00   1.268974e+00  -1.486035e+00
```

and the code for the split step Fourier method is

```
/*  split.c -- Programming Project 2
    Nonlinear Schroedinger using Split Step Fourier Method

```

```
    This is method 2(i) described in

    Thiab Taha, Mark Ablowitz, Analytical and Numerical Aspects of
    Certain Nonlinear Evolution Equations II.  Numerical, Nonlinear
    Schrodinger Equation, J. of Comp. Physics, 55 (1984), 203-230.
*/

#include <stdio.h>
#include <math.h>
#include <fftw3.h>
#include "utility.h"


Complex split(int K,Complex u[K],int N){
    Complex qt[K],qthat[K];
    fftw_plan qt2qthat,qthat2u;
    qt2qthat=fftw_plan_dft_1d(K,(fftw_complex *)qt,
        (fftw_complex *)qthat,FFTW_FORWARD,FFTW_ESTIMATE);
    qthat2u=fftw_plan_dft_1d(K,(fftw_complex *)qthat,
        (fftw_complex *)u,FFTW_BACKWARD,FFTW_ESTIMATE);
    int k;
    double rho=dt*W*W;
    int n;
    for(n=0;n<N;n++){
        int k;
        for (k=0;k<K;k++) qt[k]=cexp(-2*I*u[k]*conj(u[k])*dt)*u[k];
        fftw_execute(qt2qthat);
        for(k=0;k<K;k++){
            int j;
            if(k<K/2) j=k; else j=k-K;
            qthat[k]*=cexp(I*j*j*rho)/K;
        }
        fftw_execute(qthat2u);
    }
    fftw_destroy_plan(qthat2u);
    fftw_destroy_plan(qt2qthat);
    return u[K/2];
}

main(){
    printf("split -- Programming Project 2\n"
        "Nonlinear Schroedinger using Split Step Fourier Method\n\n");
    everyone(1024,split,800000);
}
```

with output

```
split -- Programming Project 2
Nonlinear Schroedinger using Split Step Fourier Method

fAlexander=2*cexp(3*I*sin(W*x))+sin(3*W*x)
fAnthony=cexp(I*W*x)-sin(2*W*x)+cexp(5*I*W*x)
fBrian=0.5*cexp(I*W*x)+0.5*sin(2*W*x)-2*I*cos(3*W*x)
fJordan=2*cexp(-I*W*x)-I*cos(3*W*x)
fJoseph=0.5*cexp(I*W*x)-cexp(2*I*W*x)+I*cexp(3*I*W*x)+0.5*cexp(5*I*W*x)
fKyle=I*sin(W*x)+cos(2*W*x)+cexp(-3*I*W*x)
fMasakazu=1+cexp(I*W*x)
fSarah=cexp(I*W*x)+cexp(-2*I*W*x)+cexp(3*I*W*x)
fShijie=sin(3*cos(W*x))+2*cexp(2*I*W*x)

L=10 K=1024 dx=0.00976562
T=0.2 N=800000 dt=2.5e-07

Name               |u(.,0)|       |u(.,T)|   Re(u(5,0.2))   Im(u(5,0.2))
fAlexander     4.500000e+00   4.500000e+00  -1.440323e+00  -2.924135e+00
fAnthony       2.500000e+00   2.500000e+00  -1.310387e+00   4.135818e-01
fBrian         2.375000e+00   2.375000e+00   1.850646e+00   1.965636e+00
fJordan        4.500000e+00   4.500000e+00   1.876497e+00   4.649049e-01
fJoseph        2.500000e+00   2.500000e+00  -8.731091e-01  -4.059225e-01
fKyle          2.000000e+00   2.000000e+00   1.264414e-01  -3.491812e-01
fMasakazu      2.000000e+00   2.000000e+00   2.377875e-03  -7.211771e-02
fSarah         3.000000e+00   3.000000e+00  -5.794367e-01   2.401274e-01
fShijie        4.424677e+00   4.424677e+00   1.268159e+00  -1.485840e+00
```

Note the two methods yield results consistent with Part 2.