Math/CS 467/667 Programming Project 3

Kuramoto–Sivashinsky Equation

1. Consider the pattern formation equation

$$u_t + uu_x + \nu u_{xx} + \mu u_{xxxx} = 0 \qquad \text{with} \qquad u(0,x) = u_0(x)$$

on the domain $[-1,1]$ with periodic boundary conditions. Approximate u and uu_x using discrete Fourier series as

$$u(x,t) \approx \sum_{n=-N/2+1}^{N/2} y_j(t) e^{\pi i n x} \quad \text{and} \quad (uu_x)(x,t) \approx \sum_{n=-N/2+1}^{N/2} B_n\big(y(t)\big) e^{\pi i n x}$$

where $y = (y_0, \ldots, y_{N/2}, y_{-N/2+1}, \ldots, y_{-1})$ to obtain the system of ordinary differential equations

$$\frac{dy_n}{dt} + B_n(y) - \nu\pi^2 n^2 y_n + \mu\pi^4 n^4 y_n = 0.$$

Note that B_n depends on t through y and may be computed using the subroutine developed in class for the viscous Burger equations. Write a program to integrate y_n on the interval $[0,T]$ using the split Euler scheme

$$y_{n,j+1} = \big(y_{n,j} - hB_n(y_{\cdot,j})\big) \exp(\nu\pi^2 n^2 h - \mu\pi^4 n^4 h)$$

where $y_{\cdot,j} = (y_{0,j}, \ldots, y_{N/2,j}, y_{-N/2+1,j}, \ldots, y_{-1,j})$ and $y_{n,j} \approx y_n(t_j)$ with $t_j = jh$.

Modifications of the file `burger.c` from April 18 yields the program

```
#include <math.h>
#include <complex.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/resource.h>

#include "fft.h"

complex f(complex x){
    return cos(M_PI*x)+sin(3*M_PI*x);
}

void makeB(int N,complex Y[N],complex B[N]){
    complex Z[N];
    fift(N,Y,Z);
    for(int n=0;n<N;n++){
        Z[n]=Z[n]*Z[n]/2;
    }
```

```
20      fft(N,Z,B);
21      for(int n=-N/2+1;n<N/2;n++){
22          int l;
23          if(n<0) l=n+N; else l=n;
24          B[l]=M_PI*I*n*B[l];
25      }
26      B[N/2]=0;
27  }
28
29  int main(){
30      double t=1.0;
31      double nu=0.01;
32      double mu=0.00001;
33      {
34          struct rlimit rlim={RLIM_INFINITY,RLIM_INFINITY};
35          setrlimit(RLIMIT_STACK,&rlim);
36      }
37      int N=128;
38      complex X[N],Y[N],Z[N],B[N];
39      for(int k=-N/2+1;k<=N/2;k++){
40          int l;
41          if(k<0) l=k+N;
42          else l=k;
43          X[l]=f(2.0*k/N);
44      }
45      fft(N,X,Y);
46      int J=16384;
47      double h=t/J;
48      for(int j=0;j<J;j++){
49          makeB(N,Y,B);
50          for(int n=-N/2+1;n<=N/2;n++){
51              int l;
52              if(n<0) l=n+N;
53              else l=n;
54              double pi2n2=M_PI*M_PI*n*n;
55              Y[l]=(Y[l]-h*B[l])
56                  *exp((nu-mu*pi2n2)*pi2n2*h);
57          }
58      }
59      fift(N,Y,Z);
60      printf("#%21s %22s %22s\n","x","Re(u)","Im(u)");
61      for(int n=-N/2+1;n<=N/2;n++){
62          double x=2.0/N*n;
63          int l;
```

```
64          if(n<0) l=n+N;
65          else l=n;
66          printf("%22.14e %22.14e %22.14e\n",x,Z[l]);
67      }
68      printf("# u(0,%d)=%g %g\n",Z[0]);
69      return 0;
70  }
```

Note the files `fft.c` and `fft.h` used by the above program are exactly the same as those developed in class.

2. Set $u_0(x) = \cos(\pi x) + \sin(3\pi x)$, $\mu = 0.00001$, $\nu = 0.01$, $N = 128$ and $h = T/J$ where $T = 1$ and $J = 16384$. Verify that $u(0, T) \approx 0.32$. Draw a plot of $u(x, T)$ versus x.

Output of the program in the previous section is

# x	Re(u)	Im(u)
-9.84375000000000e-01	-3.31351806656911e-01	1.11258388213050e-16
-9.68750000000000e-01	-3.16957218591146e-01	-1.21515972428935e-16
-9.53125000000000e-01	-3.02463087409271e-01	1.07346041763733e-16
-9.37500000000000e-01	-2.87953426249133e-01	-5.96681815603358e-17
-9.21875000000000e-01	-2.73478422140523e-01	1.66769539444308e-16
-9.06250000000000e-01	-2.59055685242220e-01	3.29504120510636e-16
-8.90625000000000e-01	-2.44679151592599e-01	2.07092641632399e-16
-8.75000000000000e-01	-2.30330225781809e-01	1.68917551427243e-16
-8.59375000000000e-01	-2.15987796028934e-01	2.37893201959357e-16
-8.43750000000000e-01	-2.01634714427441e-01	1.30018931587702e-16
-8.28125000000000e-01	-1.87260629233661e-01	1.14284935667640e-16
-8.12500000000000e-01	-1.72861469361261e-01	1.64141838226762e-17
-7.96875000000000e-01	-1.58437056185983e-01	6.96250247896069e-17
-7.81250000000000e-01	-1.43987763453820e-01	2.74860331017367e-16
-7.65625000000000e-01	-1.29511473234159e-01	3.75360818802149e-16
-7.50000000000000e-01	-1.15001167369109e-01	3.58149542540770e-16
-7.34375000000000e-01	-1.00443653122841e-01	6.03919855390463e-16
-7.18750000000000e-01	-8.58192024640695e-02	7.25029083847747e-16
-7.03125000000000e-01	-7.11022121698349e-02	4.05718479631744e-16
-6.87500000000000e-01	-5.62625819901779e-02	3.21970983154562e-16
-6.71875000000000e-01	-4.12679682324690e-02	3.47180780945896e-16
-6.56250000000000e-01	-2.60868325351960e-02	2.80931863183286e-16
-6.40625000000000e-01	-1.06926248436546e-02	2.27475642475127e-16
-6.25000000000000e-01	4.93080622191200e-03	1.44992438724644e-16
-6.09375000000000e-01	2.07822686530851e-02	-1.19069785813031e-17
-5.93750000000000e-01	3.68348472322639e-02	-1.56978403943498e-17
-5.78125000000000e-01	5.30263135509711e-02	-1.94495843056232e-16
-5.62500000000000e-01	6.92492274421511e-02	-1.81344292438680e-16
-5.46875000000000e-01	8.53421435631286e-02	-2.63441882597940e-16
-5.31250000000000e-01	1.01084203338290e-01	-3.76528334211924e-16
-5.15625000000000e-01	1.16195980288409e-01	-3.41079976776272e-16
-5.00000000000000e-01	1.30349859862658e-01	-4.27756502209838e-16
-4.84375000000000e-01	1.43192871546708e-01	-5.34058744850322e-16
-4.68750000000000e-01	1.54383546799256e-01	-6.07238545702441e-16
-4.53125000000000e-01	1.63641844803555e-01	-1.56331926584742e-16

```
-4.37500000000000e-01   1.70807620557024e-01  -1.84568271830666e-16
-4.21875000000000e-01   1.75899515267058e-01  -6.91528532885376e-17
-4.06250000000000e-01   1.79163604055794e-01  -4.51961503003540e-17
-3.90625000000000e-01   1.81101584984158e-01  -7.04631145238900e-17
-3.75000000000000e-01   1.82472138241455e-01   5.15862246622605e-16
-3.59375000000000e-01   1.84266458162419e-01   4.32182231268760e-16
-3.43750000000000e-01   1.87666805668354e-01   6.29619292669022e-16
-3.28125000000000e-01   1.94002475186267e-01   4.43882396103233e-16
-3.12500000000000e-01   2.04716118011815e-01   3.32133856450455e-16
-2.96875000000000e-01   2.21343557348961e-01   2.50036266291195e-16
-2.81250000000000e-01   2.45490368606730e-01   4.62210466422862e-16
-2.65625000000000e-01   2.78761349453151e-01   2.46991281579866e-16
-2.50000000000000e-01   3.22566756603715e-01   2.35431319251523e-16
-2.34375000000000e-01   3.77702158023782e-01   3.50650227897850e-16
-2.18750000000000e-01   4.43596781267447e-01   1.21345314207818e-16
-2.03125000000000e-01   5.17196085018776e-01   4.12657373535651e-16
-1.87500000000000e-01   5.91659087576547e-01   7.24426829581181e-16
-1.71875000000000e-01   6.55469925223065e-01   4.09630826081061e-16
-1.56250000000000e-01   6.93086630013680e-01  -2.70710202177339e-16
-1.40625000000000e-01   6.88412734140531e-01  -7.97745931827166e-16
-1.25000000000000e-01   6.31354587914266e-01  -1.21156131948922e-15
-1.09375000000000e-01   5.25165290151011e-01  -1.17764115443772e-15
-9.37500000000000e-02   3.89733867108123e-01  -1.09816528940388e-15
-7.81250000000000e-02   2.56780045018855e-01  -1.22145214083450e-15
-6.25000000000000e-02   1.58301887817140e-01  -6.53189077904371e-16
-4.68750000000000e-02   1.15236136453599e-01  -2.53033541742079e-16
-3.12500000000000e-02   1.33081364292119e-01   1.61235943340886e-16
-1.56250000000000e-02   2.05346848504086e-01   3.77095542278126e-16
 0.00000000000000e+00   3.20251592879676e-01   7.33015536290293e-16
 1.56250000000000e-02   4.65335165831655e-01   7.39228286516654e-16
 3.12500000000000e-02   6.27328775404055e-01   9.81768158292314e-16
 4.68750000000000e-02   7.87852809547353e-01   8.01235432154456e-16
 6.25000000000000e-02   9.18248760312551e-01   7.59121299100717e-16
 7.81250000000000e-02   9.79377432141319e-01   1.39013963828679e-16
 9.37500000000000e-02   9.32888531038331e-01  -3.64385269880086e-16
 1.09375000000000e-01   7.64479934207853e-01  -1.09741941230216e-15
 1.25000000000000e-01   5.05479138108737e-01  -1.30212795620109e-15
 1.40625000000000e-01   2.30000712027752e-01  -1.23315230566898e-15
 1.56250000000000e-01   2.07844069715985e-02  -8.41426214959310e-16
 1.71875000000000e-01  -7.17258726209691e-02  -1.04290222305438e-16
 1.87500000000000e-01  -4.50326918442712e-02   3.00908833882873e-16
 2.03125000000000e-01   6.99905997684596e-02   6.45553218813907e-16
 2.18750000000000e-01   2.33434390914167e-01   6.35682814020543e-16
 2.34375000000000e-01   4.11254926373351e-01   3.47605243186521e-16
 2.50000000000000e-01   5.76786192207050e-01   7.18972025543946e-16
 2.65625000000000e-01   7.05484234527058e-01   6.87186582237350e-16
 2.81250000000000e-01   7.71886053726617e-01   4.75228903307087e-16
 2.96875000000000e-01   7.55204914639850e-01   7.26515722441966e-17
 3.12500000000000e-01   6.53412541120 43e-01   1.83193105076417e-16
 3.28125000000000e-01   4.96142428338487e-01  -1.10786216711981e-16
 3.43750000000000e-01   3.42009132770786e-01  -4.82346466246509e-16
 3.59375000000000e-01   2.55327528317693e-01  -8.27236230918772e-16
 3.75000000000000e-01   2.74733643812349e-01  -9.92986161516141e-16
 3.90625000000000e-01   3.91586774752935e-01  -1.10131332149474e-15
```

```
  4.06250000000000e-01   5.44835000137011e-01  -6.95709442977258e-16
  4.21875000000000e-01   6.31933200479129e-01  -3.47151508942191e-16
  4.37500000000000e-01   5.43694537762018e-01   1.26194563957402e-15
  4.53125000000000e-01   2.29222152473109e-01   2.61446436404758e-15
  4.68750000000000e-01  -2.44281736967607e-01   3.30802432876281e-15
  4.84375000000000e-01  -7.09283875034358e-01   3.33653379229456e-15
  5.00000000000000e-01  -1.00550932913463e+00   1.54320952641103e-15
  5.15625000000000e-01  -1.07924267514872e+00   3.57589121801757e-16
  5.31250000000000e-01  -9.86520324643311e-01  -4.33766198104760e-16
  5.46875000000000e-01  -8.27241984486119e-01  -3.64498743701959e-16
  5.62500000000000e-01  -6.84952095594911e-01  -6.98046420719801e-16
  5.78125000000000e-01  -6.03276264545995e-01  -7.49164455871446e-16
  5.93750000000000e-01  -5.88198427626110e-01  -5.58674299189489e-16
  6.09375000000000e-01  -6.20389094105974e-01  -6.53330202452097e-16
  6.25000000000000e-01  -6.70087858895247e-01  -1.14947229031513e-15
  6.40625000000000e-01  -7.10723182591958e-01  -7.61307520203284e-16
  6.56250000000000e-01  -7.27482277142688e-01  -1.25775984919374e-15
  6.71875000000000e-01  -7.18761344800900e-01  -6.10829477290665e-16
  6.87500000000000e-01  -6.91999697274606e-01  -5.83800138865299e-16
  7.03125000000000e-01  -6.57634377158736e-01  -8.60186758333962e-16
  7.18750000000000e-01  -6.24364422201492e-01  -2.87190075199119e-16
  7.34375000000000e-01  -5.96971779494118e-01  -1.69342352654568e-16
  7.50000000000000e-01  -5.76362379665659e-01  -1.53146739367282e-16
  7.65625000000000e-01  -5.60860439591713e-01   1.00850047357189e-16
  7.81250000000000e-01  -5.47816920765327e-01  -4.51881394859555e-17
  7.96875000000000e-01  -5.34895871658251e-01   2.46123919841877e-16
  8.12500000000000e-01  -5.20751853703667e-01  -3.30285043812718e-16
  8.28125000000000e-01  -5.05113588200488e-01  -8.99695350002593e-17
  8.43750000000000e-01  -4.88473424258001e-01   1.17867856441466e-16
  8.59375000000000e-01  -4.71636663659740e-01  -3.25901146361475e-16
  8.75000000000000e-01  -4.55331032712759e-01  -3.78894051020351e-16
  8.90625000000000e-01  -4.39980211154990e-01  -2.33951583506334e-16
  9.06250000000000e-01  -4.25651823598116e-01  -5.56931564899114e-16
  9.21875000000000e-01  -4.12133629865041e-01  -1.11229116209345e-16
  9.37500000000000e-01  -3.99069258618608e-01  -1.60527610726958e-16
  9.53125000000000e-01  -3.86093361294106e-01  -3.84872525916317e-16
  9.68750000000000e-01  -3.72926723560614e-01   5.11083050297342e-18
  9.84375000000000e-01  -3.59418302504556e-01   8.91314452659763e-17
  1.00000000000000e+00  -3.45539953114715e-01  -1.14396255532534e-16
# u(0,4200959)=0.320252 7.33016e-16
```

which shows

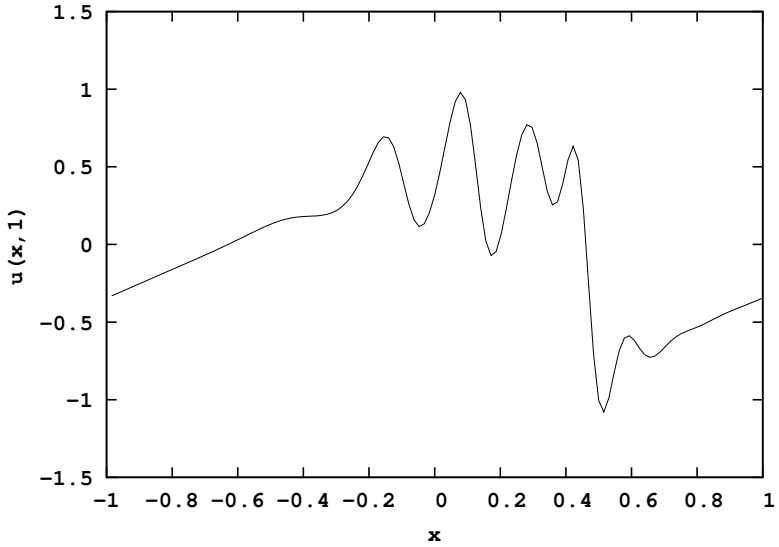
$$u(0,T) \approx 0.320252.$$

The `gnuplot` plotting script

```
set terminal postscript enhanced eps font "Courier-Bold"
set output 'plot2.eps'
set size 0.7,0.7
set xlabel "x"
set ylabel "u(x,1)"
set key left top
set key spacing 1.5
set style data lines
set sample 800
```

```
plot [] [-1.5:1.5] "p2.out" ti ""
```

creates the graph



3. For convenience define $\alpha_n = \nu\pi^2 n^2 h - \mu\pi^4 n^4 h$ and modify your code to use the split RK2 scheme given by

$$\begin{aligned} k_{1,n} &= -hB_n(y_{\cdot,j}) \\ k_{2,n} &= -he^{-\alpha_n} B_n\big(p\big) \qquad \text{where} \qquad p_n = (y_{n,j} + k_{1,n})e^{\alpha_n} \\ y_{n,j+1} &= (y_{n,j} + (k_{1,n} + k_{2,n})/2)e^{\alpha_n}. \end{aligned}$$

Let U^h be the approximation of $u(T)$ using the split RK2 method with step size h. Graph $\log \|U^h - U^{h/2}\|$ versus $\log h$ where $h = 2^{-j}$ for $j = 6, \ldots, 16$ and

$$\big\|U^h - U^{h/2}\big\| = \sqrt{\frac{2}{N} \sum_{\ell=-N/2+1}^{N/2} \left| U^h\Big(\frac{2\ell}{N}\Big) - U^{h/2}\Big(\frac{2\ell}{N}\Big) \right|^2}$$

to verify the order of convergence for the split RK2 method numerically. What happens if you take $N = 256$?

The program

```
1 #include <math.h>
2 #include <complex.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <sys/time.h>
6 #include <sys/resource.h>
```

```
7
8  #include "fft.h"
9
10 complex f(complex x){
11     return cos(M_PI*x)+sin(3*M_PI*x);
12 }
13
14 void makeB(int N,complex Y[N],complex B[N]){
15     complex Z[N];
16     fift(N,Y,Z);
17     for(int n=0;n<N;n++){
18         Z[n]=Z[n]*Z[n]/2;
19     }
20     fft(N,Z,B);
21     for(int n=-N/2+1;n<N/2;n++){
22         int l;
23         if(n<0) l=n+N; else l=n;
24         B[l]=M_PI*I*n*B[l];
25     }
26     B[N/2]=0;
27 }
28
29 double dist(int N,complex X[N],complex Y[N]){
30     double r=0;
31     for(int n=0;n<N;n++){
32         double Z=X[n]-Y[n];
33         r=r+Z*conj(Z);
34     }
35     return sqrt(r);
36 }
37
38 int main(){
39     double t=1.0;
40     double nu=0.01;
41     double mu=0.00001;
42     {
43         struct rlimit rlim={RLIM_INFINITY,RLIM_INFINITY};
44         setrlimit(RLIMIT_STACK,&rlim);
45     }
46     int N=128;
47     complex X[N],Y[N],Z[N],Zold[N],B[N];
48     for(int k=-N/2+1;k<=N/2;k++){
49         int l;
50         if(k<0) l=k+N;
```

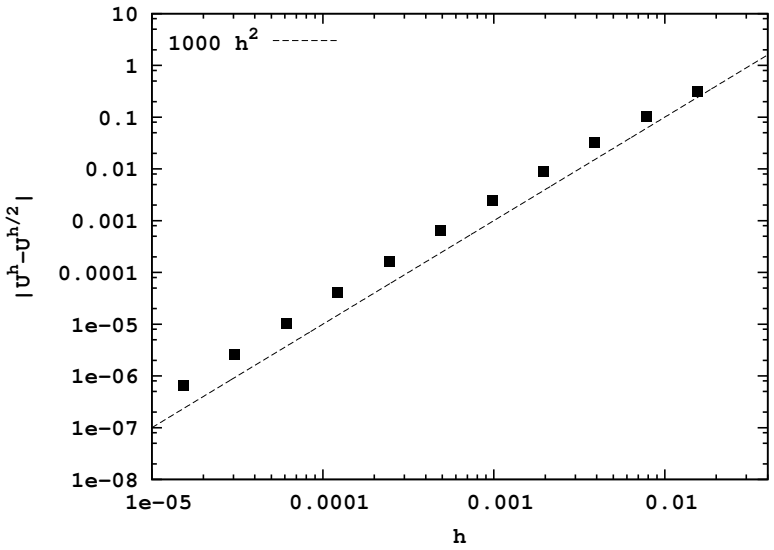
```
51          else l=k;
52          X[l]=f(2.0*k/N);
53      }
54      printf("#%7s %22s %22s %22s\n","J","|U^h-U^h/2|",
55          "Re(U^h(0,1))","Im(U^h(0,1))");
56      for(int J=64;J<=131072;J=J*2){
57          fft(N,X,Y);
58          double h=t/J;
59          for(int j=0;j<J;j++){
60              complex k1[N],k2[N],tmp[N];
61              makeB(N,Y,B);
62              for(int n=-N/2+1;n<=N/2;n++){
63                  int l=n<0?n+N:n;
64                  double pi2n2=M_PI*M_PI*n*n;
65                  double alpha=(nu-mu*pi2n2)*pi2n2*h;
66                  k1[l]=-h*B[l];
67                  tmp[l]=(Y[l]+k1[l])*exp(alpha);
68              }  // k1=h f(yn,tn)
69                 // tmp=(yn+k1)*exp(-ah)
70              makeB(N,tmp,B);
71              for(int n=-N/2+1;n<=N/2;n++){
72                  int l=n<0?n+N:n;
73                  double pi2n2=M_PI*M_PI*n*n;
74                  double alpha=(nu-mu*pi2n2)*pi2n2*h;
75                  k2[l]=-h*B[l]*exp(-alpha);
76              }  // k2=h exp(ah)f(tmp,tn+h)
77              for(int n=-N/2+1;n<=N/2;n++){
78                  int l=n<0?n+N:n;
79                  double pi2n2=M_PI*M_PI*n*n;
80                  double alpha=(nu-mu*pi2n2)*pi2n2*h;
81                  Y[l]=(Y[l]+(k1[l]+k2[l])/2)*exp(alpha);
82              }
83          }
84          fift(N,Y,Z);
85          if(J>64){
86              double er=dist(N,Z,Zold);
87              printf("%8d %22.14e %22.14e %22.14e\n",J/2,er,Zold[0]);
88          }
89          memcpy(Zold,Z,N*sizeof(complex));
90      }
91      return 0;
92  }
```

produces the output

```
#      J            |U^h-U^h/2|           Re(U^h(0,1))           Im(U^h(0,1))
```

```
   64   3.10756616134137e-01   3.62392788177117e-01   3.02926086992450e-16
  128   1.05375754534531e-01   3.32479909418346e-01   6.94485701585590e-16
  256   3.19279016825811e-02   3.25954839634281e-01   4.74202925190848e-16
  512   9.05131470037609e-03   3.24069379397074e-01   2.93195372494393e-16
 1024   2.44362594469074e-03   3.23494049558329e-01   1.60684182973214e-15
 2048   6.37757854926851e-04   3.23328738258165e-01   7.26334140402352e-16
 4096   1.63099281278662e-04   3.23283920075056e-01  -2.65277166552891e-16
 8192   4.12513197927348e-05   3.23272214231917e-01  -5.37899802824371e-16
16384   1.03734905032436e-05   3.23269220425602e-01  -5.43429233904047e-16
32768   2.60101359093303e-06   3.23268463233068e-01   1.23197892859528e-15
65536   6.51207475019377e-07   3.23268272820186e-01  -2.32832416541262e-17
```

with graph



The fact that the points agree with curve $y = Kh^2$ where $K \approx 1000$ implies the method is functioning as a second order method. We now take $N = 256$ in line 46 of the above program and to obtain

```
#      J            |U^h-U^h/2|          Re(U^h(0,1))          Im(U^h(0,1))
      64                  -nan                  -nan                  -nan
     128                  -nan                  -nan                  -nan
     256                  -nan                  -nan                  -nan
     512   1.28004919089024e-02   3.24069391502699e-01   5.63954742249886e-16
    1024   3.45580884963594e-03   3.23494071892084e-01   9.84879328972843e-16
    2048   9.01925787087746e-04   3.23328759556400e-01   3.19414670226080e-16
    4096   2.30657203074667e-04   3.23283935481253e-01   2.10888953038715e-15
    8192   5.83381658781480e-05   3.23272225673768e-01  -4.46709403701998e-16
   16384   1.46703259371521e-05   3.23269230410368e-01   1.18716435426741e-15
   32768   3.67838795416100e-06   3.23268472798772e-01   9.33917298879351e-16
   65536   9.20946369043414e-07   3.23268282275223e-01   2.27038698774333e-15
```

Note that for values of J equal to 64, 128 and 256 the computation returns *not a number* as the result. This is because larger values of N require smaller time steps in order for the method to be stable. When $J = 65536$ the value of $U^h(0, 1) \approx 3.232682$ when either

$N = 128$ or $N = 256$. This suggests that the exact solution $u(0,1)$ is equal 3.232682 to seven significant digits.

4. [Extra Credit] Repeat the previous question for the split RK4 method. Approximate the value of $u(0,T)$ with as much precision as possible by increasing J and N.

Following the handout on split RK schemes, consider a differential equation of the form

$$\frac{dy}{dt} + ay = f(y,t) \qquad \text{with} \qquad y(t_n) = y_n.$$

Introduce the variables

$$w = e^{a(t-t_n)}y \qquad \text{and} \qquad g(w,t) = e^{a(t-tn)}f(we^{-a(t-t_n)},t)$$

to obtain the differential equation

$$\frac{dw}{dt} = g(w,t) \qquad \text{with} \qquad w(t_n) = y_n.$$

The four-stage fourth-order Runge-Kutta scheme for integrating w is

$$\begin{aligned} k_1 &= hg(w_n, t_n) \\ k_2 &= hg(w_n + k_1/2, t_n + h/2) \\ k_3 &= hg(w_n + k_2/2, t_n + h/2) \\ k_4 &= hg(w_n + k_3, t_n + h) \\ w_{n+1} &= w_n + (k_1 + 2k_2 + 2k_3 + k_4)/6. \end{aligned}$$

In terms of y and f this may be written

$$\begin{aligned} k_1 &= hf(y_n, t_n) \\ k_2 &= he^{ah/2}f\big((y_n + k_1/2)e^{-ah/2}, t_n + h/2\big) \\ k_3 &= he^{ah/2}f\big((y_n + k_2/2)e^{-ah/2}, t_n + h/2\big) \\ k_4 &= he^{ah}f\big((w_n + k_3)e^{-ah}, t_n + h\big) \\ y_{n+1} &= e^{-ah}\big(y_n + (k_1 + 2k_2 + 2k_3 + k_4)/6\big). \end{aligned}$$

The resulting program

```
1 #include <math.h>
2 #include <complex.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <sys/time.h>
6 #include <sys/resource.h>
7
```

```
8  #include "fft.h"
9
10 complex f(complex x){
11     return cos(M_PI*x)+sin(3*M_PI*x);
12 }
13
14 void makeB(int N,complex Y[N],complex B[N]){
15     complex Z[N];
16     fift(N,Y,Z);
17     for(int n=0;n<N;n++){
18         Z[n]=Z[n]*Z[n]/2;
19     }
20     fft(N,Z,B);
21     for(int n=-N/2+1;n<N/2;n++){
22         int l;
23         if(n<0) l=n+N; else l=n;
24         B[l]=M_PI*I*n*B[l];
25     }
26     B[N/2]=0;
27 }
28
29 double dist(int N,complex X[N],complex Y[N]){
30     double r=0;
31     for(int n=0;n<N;n++){
32         double Z=X[n]-Y[n];
33         r=r+Z*conj(Z);
34     }
35     return sqrt(r);
36 }
37
38 int main(){
39     double t=1.0;
40     double nu=0.01;
41     double mu=0.00001;
42     {
43         struct rlimit rlim={RLIM_INFINITY,RLIM_INFINITY};
44         setrlimit(RLIMIT_STACK,&rlim);
45     }
46     int N=128;
47     complex X[N],Y[N],Z[N],Zold[N],B[N];
48     for(int k=-N/2+1;k<=N/2;k++){
49         int l;
50         if(k<0) l=k+N;
51         else l=k;
```

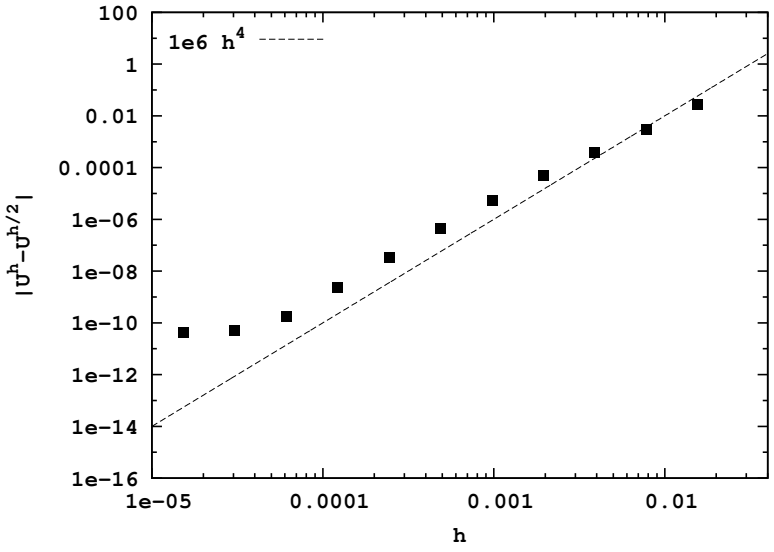
```
52          X[l]=f(2.0*k/N);
53      }
54      printf("#%7s %22s %22s %22s\n","J","|U^h-U^h/2|",
55          "Re(U^h(0,1))","Im(U^h(0,1))");
56      for(int J=64;J<=131072;J=J*2){
57          fft(N,X,Y);
58          double h=t/J;
59          for(int j=0;j<J;j++){
60              complex k1[N],k2[N],k3[N],k4[N],tmp[N];
61              makeB(N,Y,B);
62              for(int n=-N/2+1;n<=N/2;n++){
63                  int l=n<0?n+N:n;
64                  double pi2n2=M_PI*M_PI*n*n;
65                  double alpha=(nu-mu*pi2n2)*pi2n2*h;
66                  k1[l]=-h*B[l];
67                  tmp[l]=(Y[l]+k1[l]/2)*exp(alpha/2);
68              }  // k1=h f(yn,tn)
69                 // tmp=(yn+k1/2)*exp(-ah/2)
70              makeB(N,tmp,B);
71              for(int n=-N/2+1;n<=N/2;n++){
72                  int l=n<0?n+N:n;
73                  double pi2n2=M_PI*M_PI*n*n;
74                  double alpha=(nu-mu*pi2n2)*pi2n2*h;
75                  k2[l]=-h*B[l]*exp(-alpha/2);
76                  tmp[l]=(Y[l]+k2[l]/2)*exp(alpha/2);
77              }  // k2=h exp(ah)f(tmp,tn+h)
78                 // tmp=(yn+k2/2)*exp(-ah/2)
79              makeB(N,tmp,B);
80              for(int n=-N/2+1;n<=N/2;n++){
81                  int l=n<0?n+N:n;
82                  double pi2n2=M_PI*M_PI*n*n;
83                  double alpha=(nu-mu*pi2n2)*pi2n2*h;
84                  k3[l]=-h*B[l]*exp(-alpha/2);
85                  tmp[l]=(Y[l]+k3[l])*exp(alpha);
86              }  // k3=h exp(ah)f(tmp,tn+h)
87                 // tmp=(yn+k3)*exp(-ah)
88              makeB(N,tmp,B);
89              for(int n=-N/2+1;n<=N/2;n++){
90                  int l=n<0?n+N:n;
91                  double pi2n2=M_PI*M_PI*n*n;
92                  double alpha=(nu-mu*pi2n2)*pi2n2*h;
93                  k4[l]=-h*B[l]*exp(-alpha);
94              }  // k4=h exp(ah)f(tmp,tn+h)
95              for(int n=-N/2+1;n<=N/2;n++){
```

```
96                  int l=n<0?n+N:n;
97                  double pi2n2=M_PI*M_PI*n*n;
98                  double alpha=(nu-mu*pi2n2)*pi2n2*h;
99                  Y[l]=(Y[l]+(k1[l]+2*(k2[l]+k3[l])+k4[l])/6)*exp(alpha);
100             }
101         }
102         fift(N,Y,Z);
103         if(J>64){
104             double er=dist(N,Z,Zold);
105             printf("%8d %22.14e %22.14e %22.14e\n",J/2,er,Zold[0]);
106         }
107         memcpy(Zold,Z,N*sizeof(complex));
108     }
109     printf("# U(0,%g) = %22.14e %22.14e\n",t,Z[0]);
110     return 0;
111 }
```

produces the output

```
#      J            |U^h-U^h/2|           Re(U^h(0,1))           Im(U^h(0,1))
      64   2.72402815406515e-02   3.23640760157633e-01   1.03909936211011e-15
     128   2.96821538740276e-03   3.23259985624597e-01   1.89328804370281e-15
     256   3.87275914862941e-04   3.23263463508242e-01   7.95885709767297e-16
     512   4.84237267753036e-05   3.23267158775571e-01  -4.09123689787405e-16
    1024   5.18033719187727e-06   3.23268079784846e-01  -2.30420066707482e-16
    2048   4.57113411391895e-07   3.23268196440703e-01   1.56550662190613e-15
    4096   3.42276472679863e-08   3.23268208005985e-01   7.42705593206883e-16
    8192   2.29776793396823e-09   3.23268209038265e-01  -6.83318419208989e-16
   16384   1.70043261258641e-10   3.23268209114652e-01  -3.13036272250877e-16
   32768   4.96273832077227e-11   3.23268209128835e-01   1.23677652320853e-15
   65536   4.16109052696695e-11   3.23268209132976e-01   2.17225326268333e-15
# U(0,1) =   3.23268209134185e-01   4.03404523327544e-16
```

with graph



Note that the curve tracks the h^4 line between $J = 64$ and 16384 after which rounding error starts to accumulate. Without concern to the spatial resolution it appears that $u(0,1) \approx 3.2326820913$ to 11 significant digits when $N = 128$. However, this approximation needs to be independent of spatial resolution. To check this, we change the spatial resolution to $N = 256$ in line 46 as was done with the second order method. The resulting output is

```
#      J            |U^h-U^h/2|          Re(U^h(0,1))          Im(U^h(0,1))
      64                  -nan                  -nan                  -nan
     128                  -nan                  -nan                  -nan
     256                  -nan                  -nan                  -nan
     512   6.84823211629319e-05   3.23267179228170e-01   1.31846727577388e-15
    1024   7.32608119311635e-06   3.23268096975500e-01   3.98658074945931e-16
    2048   6.46163156342597e-07   3.23268208386574e-01  -3.57117867775893e-16
    4096   4.85126228290894e-08   3.23268217793079e-01   1.62561863359529e-15
    8192   3.27973601136617e-09   3.23268218487246e-01   5.73419362671052e-16
   16384   2.42941113835187e-10   3.23268218534087e-01   1.50394736591826e-15
   32768   7.05011257131135e-11   3.23268218546220e-01   7.65102539785980e-16
   65536   5.94717313770905e-11   3.23268218550231e-01   1.77506983239390e-15
# U(0,1) =   3.23268218551390e-01   6.05055373234704e-16
```

Increasing the spatial resolution now causes us to revise the estimate before to only 7 significant digits to obtain $u(0,1) \approx 3.232682$. As suggested from Programming Project 2 on the use of Fourier series to approximate derivatives, increasing spatial resolution more without using higher precision arithmetic may not increase accuracy. Therefore, finding about 7 significant digits appears all we are able to do obtain at this point even with a fourth order method.