Math 701: Project 1

1 Kuramoto-Sivashinky Equation

1. Consider the hyper-diffusion equation given by

$$\begin{cases} v_t + \nu v_{xxxx} = 0 & \text{for } x \in (0,1) \times (0,T) \\ v(0,t) = v(1,t) = 0 & \\ v_{xx}(0,t) = v_{xx}(1,t) = 0 & \text{for } t \in [0,T] \\ v(x,0) = f(x) & \text{for } x \in [0,1] \end{cases}$$

and the finite difference approximation

$$\begin{cases} u_k^{n+1} = u_k^n - \rho\delta^4 u_n^k & \text{for } k = 1, \cdots, K-1 \\ & n = 0, \cdots, N-1 \\ u_0^n = u_K^n = 0 & \\ u_{-1}^n = -u_1^n \quad \text{and} \quad u_{K+1}^n = -u_{K-1}^n & \text{for } n = 0, \cdots, N \\ u_k^0 = f(k\Delta x) & \text{for } k = 0, \cdots, K \end{cases}$$

where $\rho = \nu\Delta t/\Delta x^4$ with $\Delta t = T/N$ and $\Delta x = 1/K$.

(i) Assume the solutions v are smooth on $[0,1] \times [0,T]$ and show the finite difference approximation is consistent. Find the order of the approximation.

We expand the approximation of the time difference $u_k^{n+1} = v(x, t + \Delta t)$ using Taylor's theorem.

$$u_k^{n+1} = u_n^n + \Delta t u_t + \frac{\Delta t^2}{2} u_{tt} + O(\Delta t^3)$$

Similarly, the term

$$\delta^4 u_k = \frac{f(x+2\Delta x) - 4f(x+\Delta x) + 6f(x) - 4f(x-\Delta x) + f(x-2\Delta x)}{\Delta x^4}$$

$$\begin{aligned} \delta^4 u_k^n &= u_{k+2}^n + u_{k-2}^n - 4u_{k+1}^n + 6u_k^n - 4u_{k-1}^n + u_{k-2}^n \\ &= u_{k+2}^n + u_{k-2}^n - 4(u_{k+1}^n + u_{k-1}^n) + 6u_k^n \end{aligned}$$

Using the Taylor expansions for the spacial derivatives, we have

$$\begin{aligned}
v(x+2\Delta x,t) &= u_{k+2}^n & &= u_k^n + 2\Delta x(u_x)_k^n + 2\Delta x^2(u_{xx})_k^n + \tfrac{4\Delta x^4}{3}(u_{xxx})_k \\
&\quad +\tfrac{2}{3}\Delta x^4(u_{xxxx})_k^n + O(\Delta x^6) \\
v(x-2\Delta x,t) &= u_{k-2}^n & &= u_k^n - 2\Delta x(u_x)_k^n + 2\Delta x^2(u_{xx})_k^n - \tfrac{4\Delta x^3}{3}(u_{xxx})_k \\
&\quad +\tfrac{2}{3}\Delta x^4(u_{xxxx})_k^n + O(\Delta x^6) \\
v(x+\Delta x,t) &= u_{k+1}^n & &= u_k^n + \Delta x(u_x)_k^n + \tfrac{\Delta x^2}{2}(u_{xx})_k^n + \tfrac{\Delta x^3}{6}(u_{xxx})_k^n \\
&\quad +\tfrac{\Delta x^4}{24}(u_{xxxx})_k^n + O(\Delta x^6) \\
v(x-\Delta x,t) &= u_{k-1}^n & &= u_k^n - \Delta x(u_x)_k^n + \tfrac{\Delta x^2}{2}(u_{xx})_k^n - \tfrac{\Delta x^3}{6}(u_{xxx})_k^n \\
&\quad +\tfrac{\Delta x^4}{24}(u_{xxxx})_k^n + O(\Delta x^6)
\end{aligned}$$

We can now simplify since all our terms are in the form u_k^n. Grouping terms,

$$u_k^{n+1} = u_k^n - \nu\frac{\Delta t}{\Delta x^4}\delta^4 u_k^n$$

$$= u_k^n - \nu\frac{\Delta t}{\Delta x^4}\big(2u_k^n + 4\Delta x^2 u_{xx} + \frac{4\Delta x^4}{3}(u_{xxxx})_k^n - 4\big(2u_k^n + \Delta x^2(u_{xx})_k^n + \frac{1}{12}\Delta x^4(u_{xxxx})_k^n\big) + 6u_k^n$$
$$+O(\Delta x^6)\big)$$

$$= u_k^n - \nu\frac{\Delta t}{\Delta x^4}\big(\Delta x^4(u_{xxxx})_k^n + O(\Delta x^6)\big)$$
$$= u_k^n - \nu\Delta t(u_{xxxx})_k^n + O(\Delta x^2)$$

We can now define our truncation error, or the order of the approximation of the scheme by setting τ_k^n equal to the difference of our expanded scheme and the differential equation. At this point, we can also drop the sub and superscripts since all terms reference the same point in space and time.

$$\tau = u + \Delta t u_t + O(\Delta t^2) - u + \nu\Delta t u_{xxxx} + O(\Delta x^2)$$

$$\tau = -\nu\Delta t u_{xxxx} + O(\Delta t^2) + \nu\Delta t u_{xxxx} + O(\Delta x^2)$$
$$\tau = -\nu u_{xxxx} + O(\Delta t) + \nu u_{xxxx} + O(\Delta x^2)$$
$$\tau = O(\Delta t) + O(\Delta x^2)$$

(ii) Find conditions under which this finite difference scheme is stable and use the Lax theorem to show this method is conditionally convergent.

We need to find the condition on the amplification factor of the wave so that the decay is such that the scheme will converge. We take the discrete Fourier transform. The discrete Fourier transform reduces the PDE to an ODE (in transform space), where we can continue our analysis due to Parseval's Identity, which states that the norm of the new ODE is the same as the norm of the PDE. Note that here the notation for u does not change after the transform for convenience. The discrete Fourier transform is stated below.

$$u(\xi) = \frac{1}{\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} e^{-ik\xi} u_k \quad \text{for } k \in [-\pi, \pi]$$

By the definition of stability, we need to find a K and β such that

$$||u^{n+1}|| \leq K e^{\beta(n+1)\Delta t} ||u^0||$$

Since our initial condition indicates that $u^0 = f(0)$, we only require that the term $Ke^{\beta(n+1)\Delta t}$ be bounded for all n, $\Delta t \leq \Delta t_0$ and $\Delta x \leq \Delta x_0$. Note that we needed to first confirm that the scheme is consistent by showing that the trucation error τ goes to zero as do Δx, Δt, which we can see is true from the previous problem. We start by taking the discrete Fourier transform of both sides of the equation

$$u^{n+1}(\xi) = \frac{1}{\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} e^{-ik\xi} u_{k+1}$$

Here ξ is our amplification factor, which moderates the decay of the wave.
Let $\rho, \Delta t$, and Δx be defined as in the scheme.
Substituting in our scheme on the right-hand-side,

$$u^{n+1}(\xi) = \frac{1}{\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} e^{-ik\xi} \big(4\rho(u_{k-1}^n + u_{k+1}^n) - \rho(u_{k-2}^n + u_{k+2}^n) + (1-6\rho)u_k^n\big)$$

$$= \frac{1}{\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} e^{-ik\xi} \big(4\rho(u_{k-1}^n + u_{k+1}^n)\big) - \sum_{k=-\infty}^{\infty} e^{-ik\xi} \big(\rho(u_{k-2}^n + u_{k+2}^n)\big) + (1-6\rho) \sum_{k=-\infty}^{\infty} e^{-ik\xi} \big(u_k^n\big)$$

$$= \frac{1}{\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} e^{-ik\xi} \big(4\rho(u_{k-1}^n + u_{k+1}^n)\big) - \sum_{k=-\infty}^{\infty} e^{-ik\xi} \big(\rho(u_{k-2}^n + u_{k+2}^n)\big) + (1-6\rho)u^n(\xi)$$

Making a change of variables $m = k \pm 1$ and $l = k \pm 2$, we can see that

$$\sum_{k=-\infty}^{\infty} e^{-ik\xi} 4u_{k\pm1}^n = \sum_{k=-\infty}^{\infty} e^{-i(m\pm1)\xi} 4u_m^n = e^{\pm1i\xi} u(\xi)$$

and

$$\sum_{k=-\infty}^{\infty} e^{-ik\xi} u_{k\pm2}^n = \sum_{k=-\infty}^{\infty} e^{-i(l\pm1)\xi} u_l^n = e^{\pm2i\xi} u(\xi)$$

Now we can simplify our original transform expression.

$$u^{n+1}(\xi) = u^n(\xi)\big(4\rho(e^{-i\xi} + e^{i\xi}) - \rho(e^{-2i\xi} + e^{2i\xi}) + (1-6\rho)\big)$$

We can simplify some by canceling complex conjugates and then using trigonometry.

$$u^{n+1}(\xi) = u^n(\xi)\big(1 - 2\rho(\cos 2\xi - 4\cos\xi + 3)\big)$$

$$u^{n+1}(\xi) = u^n(\xi)\big(1 - 2\rho(\cos^2\xi - \sin^2\xi - 4\cos\xi + \sin^{\xi} + \cos^2\xi + 2)\big)$$

$$u^{n+1}(\xi) = u^n(\xi)\big(1 - 4\rho(\cos^2\xi - 2\cos\xi + 1)\big)$$

$$u^{n+1}(\xi) = u^n(\xi)\big(1 - 4\rho(\cos\xi - 1)^2\big)$$

$$u^{n+1}(\xi) = u^n(\xi)\big(1 - 4\rho(\cos\xi - \cos 0)^2\big)$$

$$u^{n+1}(\xi) = u^n(\xi)\big(1 - 4\rho(-2\sin^2\frac{\xi}{2})^2\big)$$

$$u^{n+1}(\xi) = u^n(\xi)\big(1 - 16\rho\sin^4\frac{\xi}{2}\big)$$

The term

$$p(\xi) = 1 - 16\rho\sin^4\frac{\xi}{2}$$

is called the *symbol* of the difference scheme. Applying the symbol to our transform $n+1$ times we have

$$u^{n+1}(\xi) = (1 - 16\rho\sin^4\frac{\xi}{2}^{n+1})u^0(\xi)$$

Now for $|1 - 16\rho\sin^4\frac{\xi}{2} \leq 1|$, we can choose $K = 1$ and $\beta = 0$ so that the inequality is satisfied and the scheme is consistent.
We have $\rho\max_{\xi\in[-\pi,\pi]}\{\sin^4\xi\} \leq \frac{1}{8}$, and by Lax Theorem the scheme is conditionally convergent.
Note that this is only one way to do a stability analysis. Alternatively, we can look at the eigenvalues of the matrix Q. The scheme, represented by the matrix equation, $u^{n+1} = Qu^n$, must satisfy the same inequality. You would need to find the spectral radius

$$\rho(Q) = max_i(|\lambda_i|)$$

The discrete von Neumann criterion yeilds the same results. Using the general Fourier mode, we set $u_k^{n+1} = \xi^{n+1}\omega^{jk}$ where $\omega^{jk} = e^{ijk\pi\Delta x}$ and our scheme becomes

$$\xi^{n+1}\omega^{jk} = \xi^n\omega^{jk} - \rho\xi\delta^4\omega^{jk}$$

$$= \xi^n\omega^{jk} - \rho\xi\delta^2\omega^{jk}$$

$$= u^n(\xi)\big(4\rho(e^{-i\xi} + e^{i\xi}) - \rho(e^{-2i\xi} + e^{2i\xi}) + (1 - 6\rho)\big)$$

The rest of the analysis is the same. We can start with the Fourier mode since we have a function that is continuous on $[-\pi, \pi]$ and is consistent.

(iii) Write a program which implements this finite difference scheme.

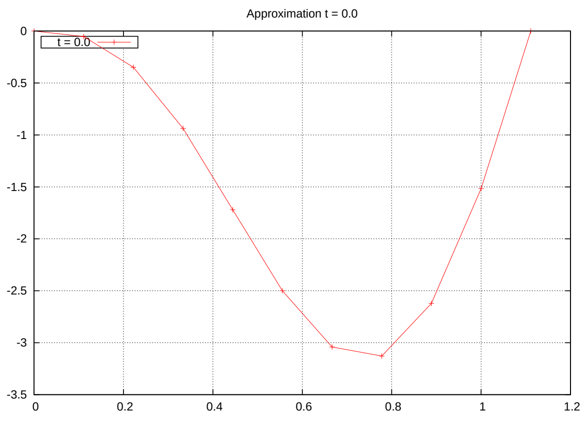
Program

```
#include "deps.h"
#include "output.h"
#include "print.h"

#define K 10
#define T 0.25
#define N 5
#define NU 0.00005
#define NMOD 1

val f(val x){
    return x*x*x*(-48.0*x*x + 112.0*x - 64.0);
}
val delta4(vec u, int i){
  if (i==0||i==K) return 0.0;
  else if (i==1) return u(3) - 4*u(2) + 5*u(1);
  else if (i==K-1) return 5*u(K-1) - 4*u(K-2) + u(K-3);
  else return u(i+2) - 4*u(i+1) + 6*u(i) - 4*u(i-1) + u(i-2);
}

main(int argc, char* argv[]){
    static vec u(K+1);
    int k,n;
    std::string op=argv[1]; // command line argument
  val dx=1.0/K, dt=T/N, dx4=dx*dx*dx*dx, xk, tn; // discretization variables

    // Initialize
    for(k=0;k<=K;k++){
        xk=k*dx;
        u(k)=f(xk);
    }
    if(op=="plot") printf("set terminal x11 noraise\nset yrange [-5:5]\nset style data
        lines\n\n");

    val rho=NU*dt/dx4; // rho = nu*dt/dx^4
    vec temp(K+1);
    temp=u;

    for(n=0;n<=N;n++){
    tn=n*dt;
        u=temp;
        for(k=0;k<=K;k++){
            xk=k*dx;
            temp(k) = u(k) − rho*delta4(u,k);
            //printu(u,tn,xk,K);
        }
        if(op=="plot0") plot0(u, tn, K−1, N);
        if(op=="plot1") plot1(u, tn, K−1, N);
        if(op=="plot3d") plot3d(u, tn, K−1, N);
        if(op=="approx") output(tn, 0.5, u(K/2), K−1, N);
    }

    return 0;
}
```

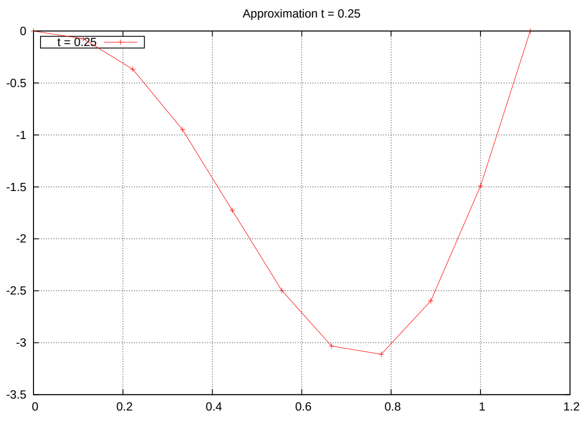
(iv) Use your program to approximate $v(x,t)$ on $[0,1] \times [0,0.25]$ where $\nu = 0.00005$ and $f(x) = -48x^5 + 112x^4 - 64x^3$. Find $v(0.5, 0.25$ to 3 significant digits.

Output

```
Approximation
    t               x               u(x,t)

0.25000         0.50000         -2.49760
```



(a) Approximate Solution at $t = 0.00$



(b) Approximate Solution at $t = 0.25$

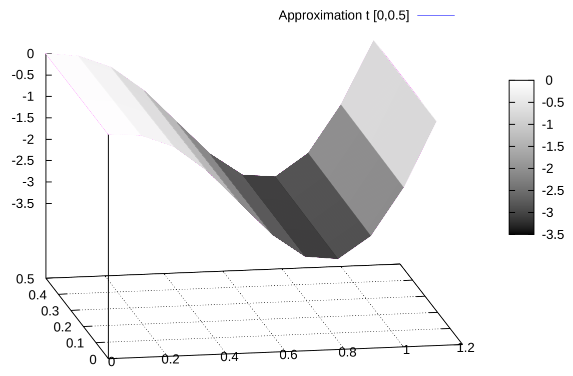


Figure 1: Approximate Solution for $t \in [0.00, 0.25]$

(v) Write a program that implements the implicit scheme

$$\begin{cases} u_k^{n+1} + \rho\delta^4 u_n^{k+1} = u_k^n & \text{for } \ k = 1, \cdots, K-1 \\ & \quad n = 0, \cdots, N-1 \\ u_0^n = u_K^n \ = 0 & \\ u_{-1}^n = -u_1^n \ \text{ and } \ u_{K+1}^n = -u_{K-1}^n & \text{for } \ n = 0, \cdots, N \\ u_k^0 = f(k\Delta x) & \text{for } \ k = 0, \cdots, K \end{cases}$$

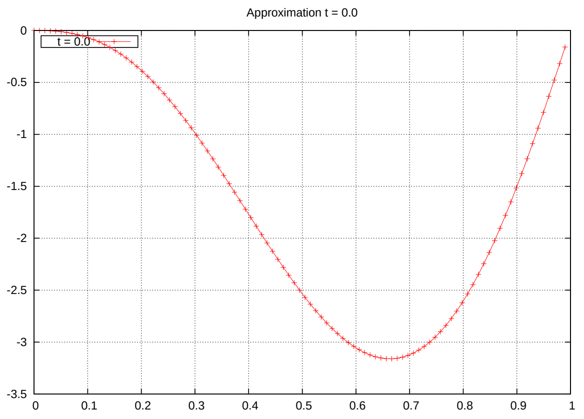
Program

```
#include "deps.h"
#include "output.h"
#include "print.h"

#define K 10
#define T 0.25
#define N 1
#define NU 0.00005
#define NMOD 200

// Note:
// Test program for gbsv used for parameters,
// followed from svn boost: https://svn.boost.org/svn/boost/sandbox/numeric_bindings/libs
    /numeric/bindings/lapack/test/ublas_gbsv.cpp

val f(val x){
    return x*x*x*(−48.0*x*x + 112.0*x − 64.0);
}

```

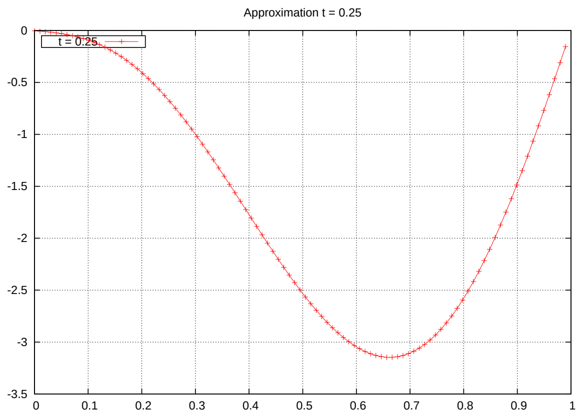
```
main(int argc, char* argv[]){
    int k,n,i,j;
    std::string op=argv[1]; // command line argument
  val dx=1.0/K, dt=T/N, dx4=dx*dx*dx*dx, xk, tn; // discretization variables
    val rho=NU*dt/dx4; // rho = nu*dt/dx^4
    static vec u(K-1), v(K+1);

    // We allocate 2 lower & 4 upper diagonal, according to the example
    static banded_matrix<val> U(K-1, K-1, 2, 4);
    vector<fortran_int_t> p(K-1);

    // Initialize matrix
    for(i=0; i<U.size1(); i++){
            U(i,i)=1.0+6.0*rho;
            k=std::max(i-1,1);
            U(k,k-1)=U(k-1,k)=-4.0*rho;
            U(k,k+1)=U(k+1,k)=-4.0*rho;
            k=std::max(i-2,2);
            U(k,k-2)=U(k-2,k)=1.0*rho;
            U(k,k+2)=U(k+2,k)=1.0*rho;
        }
    // Boundary Conditions
    U(0,0)-=1.0*rho;
    U(K-2,K-2)-=1.0*rho;
    if(op=="matrix"){ printf("Pentadiagonal Matrix\n"); matprintf(U);}

    // Initial conditions
    for(k=0;k<=K;k++){
        xk=k*dx;
        v(k)=f(xk);
    }
    u=subrange(v,1,K);
    //printf("Original Vector\n"); vecprintf(u,dx);

    lapack::gbtrf(U, p); // LU-decompostion
    for(n=0;n<=N;n++){
        tn=n*dt;
        lapack::gbtrs(U, p, u); // Solve
        if(op=="plot0") plot0(u, tn, K-1, N);
        if(op=="plot1") plot1(u, tn, K-1, N);
        if(op=="plot3d" && (n-1)%NMOD==0) plot3d(u, tn, K-1, N);
        if(op=="approx") output(tn, 0.5, u(K/2-1), K-1, N);
    }
    return 0;
}
```

(vi) Use this implicit scheme to approximate $v(0.5, 0.25)$.
Output

```
Approximation
    t                x              u(x,t)

0.25000         0.50000          -2.49759
```



(a) Approximate Solution at $t = 0.00$



(b) Approximate Solution at $t = 0.25$

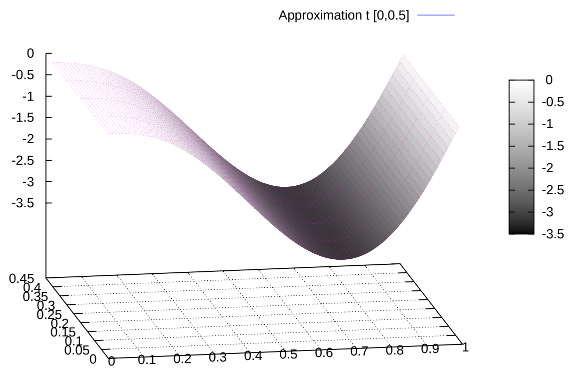


Figure 2: Approximate Solution for $t \in [0.00, 0.25]$

(vii) Numerically test whether the implicit scheme is stable for all Δx and Δt.

If we stay consistent with the restraints from the explicit scheme, then $\rho < \frac{1}{8} \rightarrow \frac{1}{2500}\Delta t < \Delta x^4$. We set $N = 5$, which will give a time step of 0.05, and then we must set $\Delta x > \frac{1}{10^4\sqrt{5}}$. We choose $\Delta x = \frac{1}{10}$. These steps are rather large for the scheme, but still statisfy the constraint.

```
Approximation
```

```
    t              x           u(x,t)

 0.25000      0.50000      -2.49712
```

We have close the same solution.
Suppose we leave the constraints of the explicit method, and choose $\Delta x = \frac{1}{100}$. We again get a good result. The scheme is stable.
Even after a single time step with a large Δx (relatively), our result is not far from the exact solution. This indicates that the value of u does not change significantly, or enough to cause the scheme to break.

2. Consider the Kuramoto-Sivashinky equation given by

$$\begin{cases} v_t + +\mu v_{xx} + \nu v_{xxxx} = 0 & \text{for } x \in (0,1) \times (0,T) \\ v(0,t) = v(1,t) = 0 & \\ v_{xx}(0,t) = v_{xx}(1,t) = 0 & \text{for } t \in [0,T] \\ v(x,0) = f(x) & \text{for } x \in [0,1] \end{cases}$$

and the finite difference approximation

$$\begin{cases} u_k^{n+1} = u_k^n - (r\delta^2 + \rho\delta^4)u_n^k - Ru_k^n\delta_0 u_k^n & \text{for } k = 1, \cdots, K-1 \\ & n = 0, \cdots, N-1 \\ u_0^n = u_K^n = 0 & \\ u_{-1}^n = -u_1^n \text{ and } u_{K+1}^n = -u_{K-1}^n & \text{for } n = 0, \cdots, N \\ u_k^0 = f(k\Delta x) & \text{for } k = 0, \cdots, K \end{cases}$$

where $\rho = \nu\Delta t \Delta x^4$ with $\Delta t = T/N$ and $\Delta x = 1/K$.

(i) Write a program which implements this finite difference scheme.

Program

```
#include "deps.h"
#include "output.h"
#include "print.h"

#define MU 0.1
#define NU 0.00005
#define K 10
#define T 0.25
#define N 100000000

val f(val x){
  return -48*pow(x,5) + 112*pow(x,4) - 64*pow(x,3);
}

val delta0(vec u, int i){
  if (i==0||i==K) return 0.0;
  return u(i+1) − u(i−1);
}

val delta2(vec u, int i){
  if (i==0||i==K) return 0.0;
  return u(i+1) − 2*u(i) + u(i−1);
}

val delta4(vec u, int i){
  if (i==0||i==K) return 0.0;
  if (i==1) return u(3) − 4*u(2) + 5*u(1);
  if (i==K−1) return 5*u(K−1) − 4*u(K−2) + u(K−3);
  return u(i+2) − 4*u(i+1) + 6*u(i) − 4*u(i−1) + u(i−2);
}

int main(){
  static vec u(K+1), un(K+1);
  val dt = T/N;
  val dx = 1.0/K;
  val gam = MU*dt/(dx*dx);
  val rho = NU*dt/(dx*dx*dx*dx);
  val ar = dt/(2*dx);
  int j,n=1;

  //Initialize U
  for (j=0;j<K+1;j++) u(j)=f(j*dx);
    //printf("Initial vector\n"); vecprintf(u,K);

  val tn = n*dt;
  while (tn<=T||n<N){
    for (j=0;j<K+1;j++) un(j) = u(j) − gam*delta2(u,j) − rho*delta4(u,j) − ar*u(j)*
        delta0(u,j);
    for (j=0;j<K+1;j++) u(j) = un(j);
    n+=1;
    tn = n*dt;
  }
  printf("At t = 0.25 and space 0.5 the 'heat' is %24.15e\n", u(K/2));
  return 0;
}
```

(ii) Use your program to approximate $v(x,t)$ on $[0,1]\times[0,0.25]$ where $\nu = 0.00005$ and $f(x) = -48x^5 + 112x^4 - 64x^3$. Find $v(0.5, 0.25$ to 2 significant digits.

(iii) Write a program that implements the semi-implicit scheme

$$\begin{cases} u_k^{n+1} & + \quad (r\delta^2 + \rho\delta^4) u_n^{k+1} = u_k^n - R u_k^n \delta_0 u_k^n & \text{for} \quad k = 1, \cdots, K-1 \\ & \qquad\qquad\qquad\qquad\qquad\qquad n = 0, \cdots, N-1 \\ u_0^n & = \quad u_K^n = 0 & \\ u_{-1}^n & = \quad -u_1^n \quad \text{and} \quad u_{K+1}^n = -u_{K-1}^n & \text{for} \quad n = 0, \cdots, N \\ u_k^0 & = \quad f(k\Delta x) & \text{for} \quad k = 0, \cdots, K \end{cases}$$

Program

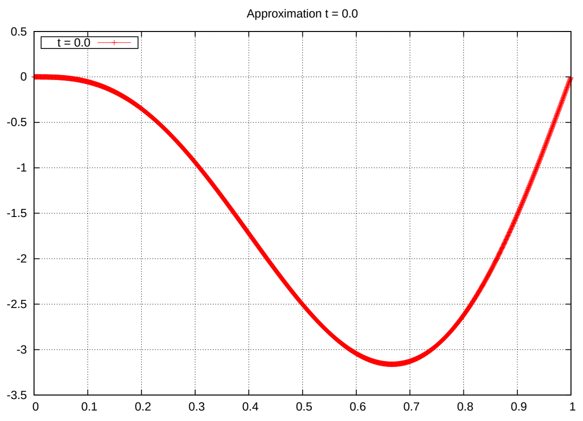
```
#include "deps.h"
#include "output.h"
#include "print.h"

#define MU 0.1
#define NU 0.00005
#define K 1000
#define T 0.25
#define N 10000

// Function for the initial condition
val f(val x){
  return x*x*x*(-48*x*x + 112*x - 64);
}

int main(){
  // Variable declaration
  static vec v(K-1), vt(K-1), u(K+1);
  val dt = T/N;
  val dx = 1.0/K;
  val gam = MU*dt/(dx*dx);
  val rho = NU*dt/(dx*dx*dx*dx);
  val ar = dt/(2*dx);
  int i,j,k;

  //Initialize U
  for (j=0;j<K-1;j++) v(j)=f((j+1)*dx);

// printf("Initial Vector\n");vecprintf(v,dx);
// printf("set terminal x11 noraise\nset yrange [-4:4]\nset style data lines\n\n");

    static banded_matrix<val, row_major> A(K-1, K-1, 2, 4), B(K-1, K-1, 2, 4);

    // Initialize matrix
    for(i=0; i< A.size1(); i++){
            A(i,i)=1.0+6.0*rho-2*gam;
            k=std::max(i-1,1);
            A(k,k+1)=A(k+1,k)=A(k,k-1)=A(k-1,k)=-4.0*rho+gam;
            k=std::max(i-2,2);
```

```
                A(k,k+2)=A(k+2,k)=A(k,k−2)=A(k−2,k)=1.0∗rho;
    }
      // Boundary Conditions
      A(0,0)−=1.0∗rho;
      A(K−2,K−2)−=1.0∗rho;
      //printf("Pentadiagonal Matrix\n"); matprintf(A);

    //Creating a permutation storage vector for the lapack functions
    vector<fortran_int_t> p(K−1);
    lapack::gbtrf(A, p); // LU−decompostion
    //matprintf(A);

    //printf("tn=%24.15e %24.15e\n",0.0,v(K/2−1));
    int n;
    for(n=1;n<=N;n++){
      val tn=n∗dt;
          //Copies the data from v into vn to be altered by lapack
      for (i=1;i<K−2;i++) vt(i) = v(i)∗(1 − ar∗(v(i+1)−v(i−1)));
      vt(0) = v(0)∗(1 − ar∗v(1));
      vt(K−2) = v(K−2)∗(1 + ar∗v(K−3));

          //printf("Interating on V\n");vecprintf(v,dx);
      B=A;
      lapack::gbtrs(B, p, vt); // Solve
            v=vt;
      //printf("tn=%24.15e %24.15e\n",tn,v(K/2−1));
    }

    //printf("tn=%24.15e %24.15e\n",0.25,v(K/2−1));
    //Reassigns the data from the lapack vector v to the spacial vector u
    for (i=1;i<K;i++) u(i)=v(i−1);
    u(0)=u(K)=0.0;

    //Prints the desired record
    printf("At x=0.5 and t=0.25 the heat is %24.15e\n",u(K/2));
    return 0;
  }
```

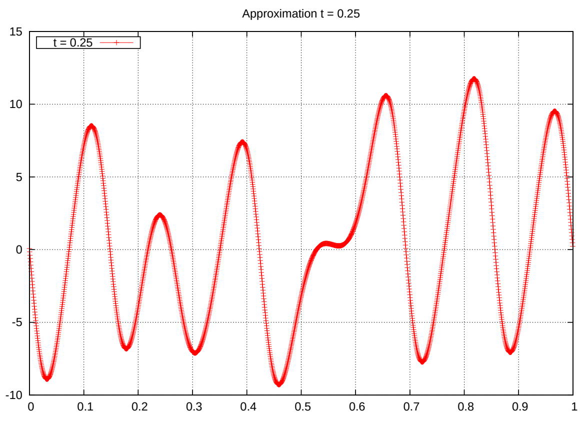
(iv) Use this semi-implicit scheme to approximate $v(0.5, 0.25)$.

Output

```
Approximation
   t            x            u(x,t)

0.25000      0.50000      -3.22645
```



(a) Approximate Solution at $t = 0.00$



(b) Exact Solution at $t = 0.25$

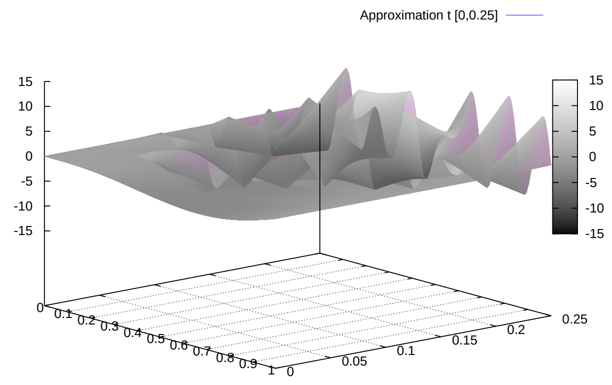


Figure 3: Approximate Solution for $t \in [0.00, 0.25]$

(v) Write a program that implements the split-implicit scheme

$$\begin{cases} u_k^{n+1} & + & (r\delta^2 + \rho\delta^4 + Ru_k^n\delta_0)u_n^{k+1} = u_k^n & \text{for } k = 1, \cdots, K-1 \\ & & & n = 0, \cdots, N-1 \\ u_0^n & = & u_K^n = 0 & \\ u_{-1}^n & = & -u_1^n \text{ and } u_{K+1}^n = -u_{K-1}^n & \text{for } n = 0, \cdots, N \\ u_k^0 & = & f(k\Delta x) & \text{for } k = 0, \cdots, K \end{cases}$$

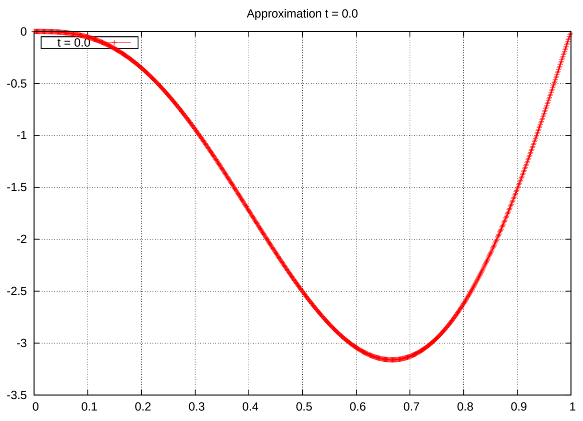
Program

```
#include "deps.h"
#include "output.h"
```

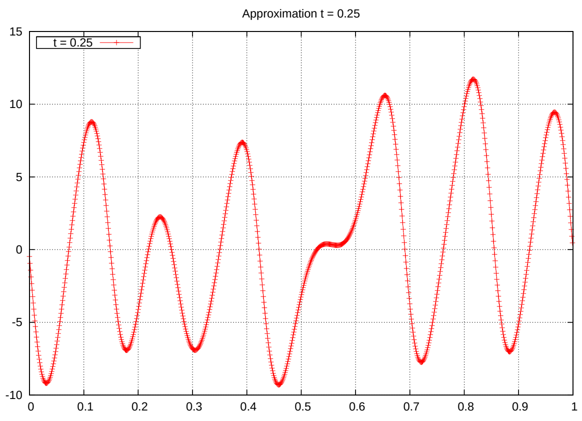
```
#include "print.h"

#define MU 0.1
#define NU 0.00005
#define K 1000
#define T 0.25
#define N 100000
#define NMOD1 1000
#define NMOD2 2000

// Function for the initial condition
val f(val x){
  return x*x*x*(-48*x*x + 112*x - 64);
}

int main(int argc, char* argv[]){
    std::string op=argv[1]; // command line argument
  // Variable declaration
  static vec v(K-1), u(K+1);
  val dt = T/N;
  val dx = 1.0/K;
  val gam = MU*dt/(dx*dx);
  val rho = NU*dt/(dx*dx*dx*dx);
  val ar = dt/(2*dx);
  int i,j,k;

  //Initialize U
  for (j=0;j<K-1;j++) v(j)=f((j+1)*dx);

    //printf("Initial Vector\n");vecprintf(v,dx);
    //printf("set terminal x11 noraise\nset yrange [-4:4]\nset style data lines\n\n");

    static banded_matrix<val, row_major> A(K-1, K-1, 2, 4);

  //Creating a permutation storage vector for the lapack functions

// printf("tn=%3g , %3g\n",0.0,v(K/2-1));
  int n;
  for(n=0;n<=N;n++){
    val tn=n*dt;

      // Initialize matrix
      for(i=0; i<A.size1(); i++){
              A(i,i)=1.0-2*gam+6*rho;
              if(i+1<A.size1())A(i,i+1)=gam-4*rho+ar*v(i);
      if(i-1>=0)A(i,i-1)=gam-4*rho-ar*v(i);
              if(i+2<A.size1())A(i,i+2)=A(i+2,i)=rho;
      if(i-2>=0)A(i,i-2)=A(i-2,i)=rho;
    }
     // Boundary Conditions
       A(0,0)=A(K-2,K-2)=1.0-2*gam+5*rho;
       //printf("Pentadiagonal Matrix\n"); matprintf(A);

    vector<fortran_int_t> p(K-1);

    lapack::gbtrf(A, p); // LU-decompostion
    //matprintf(A);

    lapack::gbtrs(A, p, v); // Solve
    //printf("tn=%24.15e %24.15e\n",tn,v(K/2-1));

        if(op=="plot0" && n%NMOD1==0) plot0(v, tn, K-1, N);
        if(op=="plot1" && n%NMOD1==0) plot1(v, tn, K-1, N);
        if(op=="plot3d" && n%NMOD2==0) plot3d(v, tn, K-1, N);
        if(op=="approx") output(tn, 0.5, v(K/2-1), K-1, N);
  }

  //Reassigns the data from the lapack vector v to the spacial vector u
  for (i=1;i<K;i++) u(i)=v(i-1);
  u(0)=u(K)=0.0;
    //Prints the desired record
  //printf("At x=0.5 and t=0.25 the heat is %24.15e\n",u(K/2));
  return 0;
}
```

(vi) Use this split-implicit scheme to approximate $v(0.5, 0.25)$.
Output

```
Approximation
   t            x            u(x,t)

0.25000      0.50000      -3.25863
```



(a) Approximate Solution at $t = 0.00$



(b) Exact Solution at $t = 0.25$

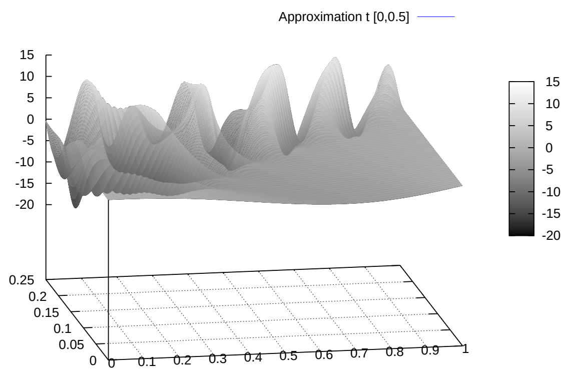


Figure 4: Approximate Solution at $t \in [0.00, 0.25]$

(vii) Compare and contrast the performance of the explicit scheme, the semi-implicit scheme and the split-implicit scheme for values of $K \in \{10, 100, 1000\}$. Which method do you like the best? Why?

We use the numbers that have been found to work for each scheme. Changing only the value of K.

Explicit Scheme

(not included)

Semi-Implicit Scheme

(Note: For comparison, the factor of 2 has been kept to preserve the ratio for stability found by numerical experimentation) $K = 10, 20$

$K = 100$

```
Approximation
   t            x            u(x,t)

0.25000      0.50000      -5.47231
```

$K = 200$

```
Approximation
   t            x            u(x,t)

0.25000      0.50000      -3.67689
```

$K = 1000$

```
Approximation
   t            x            u(x,t)

0.25000      0.50000      -3.22973
```

Split-Implicit Scheme
$K = 10$

```
Approximation
   t            x            u(x,t)

0.25000      0.50000      -33.68579
```

$K = 100$

```
Approximation
   t            x            u(x,t)

0.25000      0.50000      -5.65546
```

$K = 1000$

```
Approximation
   t            x            u(x,t)

0.25000      0.50000      -3.25863
```

If we are considering speed of computation, the best method is the semi-implicit scheme. It is not as stable as the split-implicit scheme for smaller values of K, and does even require more spacial steps for improved accuraccy, it is faster at computing a reasonable solution.
If we are considering stability, then the split-implicit scheme is the best, and will attempt to converge to a solution even with unreasonable values of K.
Saying this, it can be asked whether or not a stable scheme that converges to an inaccurate solution for bad time or spacial steps is better than one that diverges except for under the appropriate conditions. It may in fact be better to have a scheme that diverges unless it converges to the correct answer.
The explicit scheme is not mentioned, as it takes to long for a reasonable number numerical tests.

Program Files:

Makefile

```
UTIL=../util

NC=−I/usr/local/include −I/usr/local/include/boost−numeric−bindings −I$(UTIL)
LIB=−L/usr/local/lib −L/usr/local/lib/boost−numeric−bindings −L/usr/lib/atlas−base
CFLAGS = −lm
LIBS= −lblas −latlas −llapack −lgfortran

OUT=.

all: splitimp

output: data

splitimp: splitimp.cpp $(UTIL)/print.h $(UTIL)/output.h $(UTIL)/deps.h
  g++ $(NC) −g $(CFLAGS) splitimp.cpp −o $(OUT)/splitimp $(LIBS) $(LIB)

data: splitimp
  ./splitimp approx > $(OUT)/splitimp.out
  ./splitimp plot0 > $(OUT)/plot0.dat
  ./splitimp plot1 > $(OUT)/plot1.dat
  ./splitimp plot3d > $(OUT)/plot3d.dat
  gnuplot plot
  gnuplot plot3d

clean:
  rm −f splitimp splitimp.out splitimp.dat matrix.out
```

Output.h

```
#ifndef OUTPUT_H
#define OUTPUT_H

#include "deps.h"
#include "print.h"

void output(val tn, val xk, val u, int K, int N){

    if((tn>0.249999 & tn<0.250001) && (xk<0.5001 && xk>0.4999)){
        printf("Approximation \n");
        printf("%2s t %10.5s x %9.5s u(x,t)\n\n"," "," "," ");
        printf("%2.5f %4s %2.5f %4s %2.5f\n",tn, " ", xk, " ", u);
    }
}
```

```
void plot0(vec u, val tn, int K, int N){

    if(tn==0.0) plotu(u, tn, K);
}
void plot1(vec u, val tn, int K, int N){

    if(tn==0.25) plotu(u, tn, K);
}
void plot3d(vec u, val tn, int K, int N){

    plotu3d(u, tn, K);
}
#endif
```

Print.h

```
#ifndef PRINT_H
#define PRINT_H

#include "deps.h"


// Vector print
void printu(vec u, val tn, val xk, int K){
    int k;
    val xkn;
    printf("%2s t %10.5s x %9.5s u(x,t)\n\n"," "," "," ");
    for(k=0;k<=K;k++){
        xkn=k*(1.0/K);
            printf("%2.5f %4s %2.5f %4s %2.5f\n",tn, " ", xk, " ", u(k));
    }
    printf("\n");
}
void vecprintf(vec u, val dx){
    int i,m;
    for(i=0;i<u.size();i++) printf("%24.15e %24.15e\n",i*dx, u(i));
    printf("\n");
}
void plotu(vec u, val tn, int K){
    int i;
    val dx=1.0/K;
    printf("\n#tn = %24.15e\n", tn);
    for(i=0;i<u.size();i++) printf("%24.15e %24.15e\n",i*dx, u(i));
}
void plotu3d(vec u, val tn, int K){
    int k;
    val dx=1.0/K;
    for(k=0;k<u.size();k++){
        val xk=k*dx;
```

```
            printf("%24.15e %24.15e %24.15e\n",tn, xk,u(k));
        }
        printf("\n");
}
// Matrix print
void printU(mat U, val tn, val xk){
        int i,m;
        printf("%2s t %10.5s x %9.5s u(x,t)\n\n"," "," "," ");
        printf("%2.5f %4s %2.5f %4s %2.5f\n",tn, " ", xk, " ", U(xk,tn));
        printf("\n");
}
void matprintrow(vec u){
        int i,m;
        if(u.size()>10) m=10;
    else m=u.size();
    for(i=0;i<m;i++) printf("%12.8f",u(i));
        if(m<u.size()) printf(" ...");
        printf("\n");
}
void matprintf(mat A){
    int i,m;
    if(A.size1()>10) m=10;
    else m=A.size1();
    for(i=0;i<m;i++){ matrix_row<mat> row(A,i); matprintrow(row);}
        if(m<A.size1()) printf(" .\n .\n .\n");
        printf("\n");
}


#endif
```

GNUPlot Files

```
set term post
set key left box
set grid
set style data linespoints
set title "Approximation t = 0.0"
set output 'plot0.eps'
plot 'plot0.dat' using 1:2 index 0 title "t = 0.0" pt 1 lc 1
set title "Approximation t = 0.25"
set output 'plot1.eps'
plot 'plot1.dat' using 1:2 index 0 title "t = 0.25" pt 1 lc 1
```

```
set term post eps
set output 'plot3d.eps'
set view 110,100
set grid
set pm3d at s hidden3d 4
set style data pm3d
splot 'plot3d.dat' using 1:2:3 with lines lc 3 title "Approximation t [0,0.5]"
```