# 1 差分法ゼミ纏め

## 読みやすさアップ

しけたい４　(永琳カラー復活？)

---

## 1　八雲が 3 人で 24 時間（いつもより長いネタ）



らんらんる～♪

式紙は契約者が居ないと、つい、やっちゃうんだ
みんなもいっしょにやってみよう♪
（服に手をかけつつ）

（隙間から現れ鼻に靴下を押し付ける）

がくっ。

♪いつでもそこに～　♪境界商事～（がくがく）

今年も始まりました 24 時間テレビです。今年も笑いあり、涙あり、アクションアリの感動の 24 時間をお届けします。24 時間の番組表は以下のようになっています♪

（素肌を番組表で隠しながら）
・吸血鬼一味による体を張ったバラエティー！
・今年もやりますマラソン！
・人気バラエティー番組が豪華ゲストで出陣！
・隙間妖怪によるチャリティー！
超・豪・華！（番組表に手をかけながら

（ぴちゅーん

チャリティーは貧乏巫女の神社の賽銭箱の**隣の賽銭箱**で行っております。チャリティーで溜まったお金はちゃｋ・・・・・・
世のため人のために有効に活用します♪

それでは、今回のマラソンランナーの紹介です！

## 2　背水の走（はいすいのらん）

・・・・・・・・・

私・・・ですか？

今年はくじ引きで負けたみたいなのよ。私達が肉体派の番組担当みたいだったから。ま、貴女がランナーってことで。某掲示板とかでさぼってないか見張られてるかもしれないから、バスとか乗っちゃ駄目よ？

えー。
私走るのって、体力勝負の中でも特に苦手なんですよ。

だって、走ると肩こるし・・・・

え、肩なんてこるかしら？

・・・・・・・（ナイフを構えて）

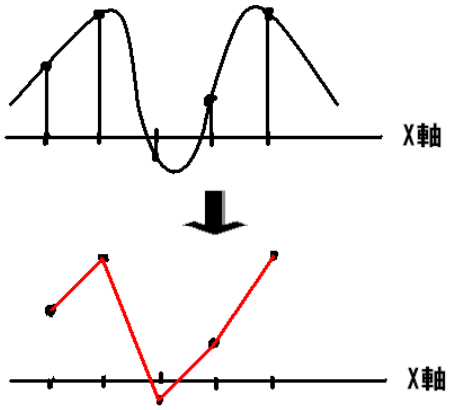
ぎゃあああああああああ（全力ダッシュ）

おおっと、これは世界新記録が更新されるかもしれません。飛んでくるナイフの軌跡を偏微分シミュレートしながらの猛ダッシュ。これは凄いです！

# 3　分点と差分近似

計算機シミュレートの目的の一つは適当な**分点**における関数の近似値を求めることです。わかりやすい問題の一つに**時間発展の偏微分方程式**があります。これは適当な**初期値**、**境界条件**、**偏微分方程式**が与えられた状態での時間発展を求めようとします。

ところが、計算機に関数の形をそのまま覚えさせるのは難しいので、いくつかの**分点**での値を覚えさせます。例えば、$f(x,t)$ の初期値として $f(x,0)=f(x)$ が得られた時、$f(0), f(0.1), f(0.2)$・・と覚えさせていきます。
時間発展のシミュレートをしたい場合は、最も単純な場合だと、後述のように偏微分方程式を適当な形で近似して、$f(0,dt), f(0.1,dt)$・・・を求めていきます。




関数の初期値を計算機に覚えさせる例。分割幅が粗いと悲惨な形になる。

さて、領域の分割の方法なのですが、ここでは $x$ の幅も $t$ の幅も均一[*1]とします。ここからの解説では $x$ の刻み幅を $h$、t の刻み幅を $dt$ としましょう。また、計算機に求めさせた関数の近似値を関数に添え字をつけて表現します。

$u(x_i, t_j)$ の近似値：$u_i^j$

$f(x_i, t_j)$ の近似値：$f_i^j$

# 4 陽的スキームの実践

折角なのでイキナリ実践してみましょう。こんな問題を考えます。ようは、ヘルムホルツの熱伝導方程式ですね

偏微分方程式：$\lambda \frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}$

境界値：$u(0,t) = \alpha, u(a,t) = \beta$

初期値：$u(x,0) = f(x)$ $x$ の定義領域：$[a,b]$

定義領域を N 等分すると、初期値が与えられていることから

$u_j^0$ $(j = 0, 1, \cdot\cdot\cdot, N+1)$

が得られます。

**重要：この教材では境界の添え字は $0, N+1$ になるように分割をします**

さて、ここである関数の偏微分

$\frac{\partial f}{\partial x}$

を**近似したい**としましょう。どうやればよいでしょうか？

答えは自己啓発系の本にありそうな文章ですが『**時と場合によります**』

[*1] 曲線を境界とした流体の問題などで用いることがある。かっこいい予感が漂うことは確かだが、適当な分割方法を決める労力、決めるときの誤差、応用性を考えると必ずしも優れているとは言えないらしい。このゼミでは扱わない。

例えば
$\frac{f_{j+1}-f_j}{h}$
は偏微分の近似に用いることが出来ます。テーラー展開で確かめてみましょう。
$f(x_{j+1}) = f(x_j) + hf'(x_j) + \frac{1}{2}h^2 f''(x_j) +$・・・・より
$\frac{f_{j+1}-f_j}{h} = f'(x_j) + \frac{1}{2}hf''(x_j) +$・・・・

となりますね。これは一次精度の近似といえます。他にも $\frac{\partial f}{\partial x}$ の表記方法には
$\frac{f_{j+1}-fj-1}{2h}$
などがあります。こちらは二次の精度です。気になる人は自分でテーラー展開するとよいでしょう。この他にもマニアックな表記方法はまだまだあります。微分される関数が滑らかならテーラー展開で精度を見ることが出来ます。テーラー展開による精度の考察は、今後も幅広く使う概念なので覚えて置いてください。

以上を踏まえて差分解法を完成させましょう。
求めたいのは $t = dt$ の時の関数の分点の近似値、$u_j^1$ $(j = 0, 1,$・・・$, N+1)$[*2]です。
未知数は $N+2$ 個で境界条件の式が二つあります。よって、あと N 個の条件式が求められれば求められそうですね。ここで差分近似
$\frac{\partial u(x_i,t_j)}{\partial t} = \frac{u_i^{j+1}-u_i^j}{dt}$
$\frac{\partial^2 u(x_i,t_j)}{\partial^2 x} = \frac{u_{i+1}^j-2u_i^j+u_{i-1}^j}{h^2}$

を用いましょう。テーラー展開をすれば解るとおり、上の式は一次、下の式は二次の精度を持っています。これを元に偏微分方程式を書き換えると
$\frac{1}{dt}(u_i^{j+1} - u_i^j) = \lambda\frac{1}{h^2}(u_{i+1}^j - 2u_i^j + u_{i-1}^j)$・・・★
となります。偏微分の定義から $i$ は $1, 2,$・・$, N$ までの値を取ることができるので、全部で $N$ 個の式が得られます。こうして、$u_j^1$ が求めることが出来るますね。

この後も、時間発展の偏微分方程式を有限要素法やＣＩＰ法などを学習していきますが、基本的な方針はこれと同じです。つまり、ある時間の分点の情報が与えられていて、その時間から微小時間後の時間発展を境界条件や、偏微分方程式の近似であらわし、微小時間後の分点の情報を作り出すというわけです。

今後の学習のために、★を書き換えて
$u_i^{j+1} = u_i^j + \gamma(u_{i+1}^j - 2u_i^j + u_{i-1}^j)$
としましょう。$\gamma = \frac{\lambda dt}{h^2}$ です。
さて、これらの各々の式は未知数が $u_i^{j+1}$ だけなので求めたい $u_i^{j+1}$ 楽々求めることが出来ます。このように連立方程式を解かなくても解ける問題を**陽的スキーム**と呼びます。精度には不安のある方法ですが、ラクチンさは群を抜いているので是非マスターするとよいでしょう！

[*2] ようは $f(x_0, t_1), f(x_1, t_1)$・・・の近似値。ここは大事なのでくどく。この部分のよい説明方法が思いついた方はしけ４まで。

# 5 陰的スキームの紹介

さて、陽的スキームの時間の偏微分の近似
$\frac{\partial u(x_i,t_j)}{\partial t} = \frac{u_i^{j+1}-u_i^j}{dt}$
ですが、実はこれは
$\frac{\partial u(x_i,t_{j+1})}{\partial t} = \frac{u_i^{j+1}-u_i^j}{dt}$

としても一次の精度であることが解るでしょうか。この近似を用いてヘルムホルツの偏微分方程式を書き直すと
$u_i^{j+1} - u_i^j = \gamma(u_{i+1}^{j+1} - 2u_i^{j+1} + u_{i-1}^{j+1})$
という式になります。式を見れば直ぐわかるとおり、この計算は一筋縄ではいきません。ですが、得られる式の数は境界条件と合わせてやっぱり $N+2$ なので**しっかり解くことができます**。

今回のは見ての通り一次の連立方程式なので逆行列を求めれば解が得られます。折角なので行列の形に直すと下のようになります。行列は $N$ **行、**$N+2$ **列**になっています。

$$\begin{bmatrix} -\gamma & 1+2\gamma & -\gamma & \cdot\cdot & \cdot\cdot & 0 \\ 0 & -\gamma & 1+2\gamma & -\gamma & \cdot\cdot & 0 \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ 0 & \cdot\cdot & \cdot\cdot & -\gamma & 1+2\gamma & -\gamma \end{bmatrix} \begin{bmatrix} u_0^{j+1} \\ u_1^{j+1} \\ \cdot\cdot \\ u_n^{j+1} \\ u_{n+1}^{j+1} \end{bmatrix} = \begin{bmatrix} u_1^j \\ u_2^j \\ \cdot\cdot \\ u_{n-1}^j \\ u_n^j \end{bmatrix}$$

$u_0^{j+1}, u_{n+1}^{j+1}$ は境界条件から求められるので、行列を N 行 N 列に書き直すことが出来ます。結果は

$$\begin{bmatrix} 1+2\gamma & -\gamma & 0 & \cdot\cdot & 0 \\ -\gamma & 1+2\gamma & -\gamma & \cdot\cdot & 0 \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ 0 & \cdot\cdot & -\gamma & 1+2\gamma & -\gamma \\ 0 & \cdot\cdot & 0 & -\gamma & 1+2\gamma \end{bmatrix} \begin{bmatrix} u_1^{j+1} \\ u_2^{j+1} \\ \cdot\cdot \\ u_{n-1}^{j+1} \\ u_n^{j+1} \end{bmatrix} = \begin{bmatrix} u_1^j + \gamma u_0^{j+1} \\ u_2^j \\ \cdot\cdot \\ u_{n-1}^j \\ u_n^j + \gamma u_{n+1}^{j+1} \end{bmatrix}$$

となります。逆行列を求めれば結果が得られますね。逆行列の解き方は番組が進めば紹介されるはずです。ちょっともどかしいかもしれませんが、我慢してくださいね♪

はぁ、はぁ。肩がこってきたよぉ・・・

### 5.1　コラム–先ほどの問の陽的スキームでの精度を上げる

（まったく疲れた様子を見せず）
精度についてちょっとだけコラムを出しておきましょうか。
$\frac{1}{\delta t}(u_i^{j+1}-u_i^j)-\lambda\frac{1}{h^2}(u_{i+1}^j-2u_i^j+u_{i-1}^j)$
の誤差はどうやって求めればいいかが今回の問題です。
テーラー展開するだけなのですけどね。

すべての展開を $(x_i,t_j)$ 周りでやるのがミソです。
で、得られた結果は
$\frac{1}{\delta t}(u_i^{j+1}-u_i^j)-\lambda\frac{1}{h^2}(u_{i+1}^j-2u_i^j+u_{i-1}^j)=\frac{\partial u(x_i,t_j)}{\partial t}-\lambda\frac{\partial^2 u(x_i,t_j)}{\partial x^2}+\frac{\delta t}{2}\frac{\partial^2 u(x_i,t_j)}{\partial t^2}-\lambda\frac{h^2}{12}\frac{\partial^4 u(x_i,t_j)}{\partial x^4}+O(\delta t^2)+O(h^4)$
ですね。

ここからちょっとトリッキーな手を使います。
問で与えられた条件より演算子に対して
$\frac{\partial}{\partial t}=\lambda\frac{\partial^2}{\partial x^2}$
が成り立つのですから、
$\frac{\partial u(x_i,t_j)}{\partial t}-\lambda\frac{\partial^2 u(x_i,t_j)}{\partial x^2}-\frac{\lambda h^2}{2}(\gamma-\frac{1}{6})\frac{\partial^4 u(x_i,t_j)}{\partial x^4}+O(\delta t^2)+O(h^4)$
となります。ここから $\gamma=\frac{1}{6}$ とすると精度のさらなる上昇が期待できることがわかるでしょう。

最後に実践演習があるので色々試してみるとよいでしょう。$\gamma$ の重要さは身に染みてわかるはずです。さて、陽的解法だけで私のナイフが見切れるかしら・・・・

## 6　最大値問題

ふふ、単純な陽的スキームでは正しくナイフの軌道を読むことは出来ないわ。うふふってどこかの魔法使いの闇歴史の一つよね。。。

ヘルムホルツの熱伝導方程式に従うナイフの軌道は私には恐れるに足りません。熱伝導の式を初期値を $\sin(\pi x)$ ！
$u(0,0)=u(1,0)=0$ ！
$\lambda=1$ ！
$h=\delta t=0.1$ ！
で計算機シミュレート！　って、あれ、負の温度が出てきたあああ。

この問いへの回答が**最大値問題**です。分点の取り方や近似方法によっては積み重なった誤差が発散してしまうという問題です。値が大きくなってしまうかどうかは偏微分方程式を積分の形になおしたり、実際に誤差をおいて考えてみても解るようですが、ここでは省略して、最も汎用性の高い**スペクトル分解**による議論ついて紹介しましょう。

これは計算機で求められた近似解を
$u_i^j = \sum_\beta \alpha_\beta^j e^{i\beta x_i}$
の形に分解することからスタートします。おなじみのフーリエ展開ですね。さて、任意の初期状態に対して時間発展に対して値が発散しないための条件は
$|\frac{\alpha_\beta^{j+1}}{\alpha_\beta^j}| < 1$
が任意の $\beta$ について満たされれば良いということが解りますね。$\alpha$ が $i$ によらないことがポイントですよ。

陽的スキームでは
$u_i^{j+1} = u_i^j + \gamma(u_{i+1}^j - 2u_i^j + u_{i-1}^j)$
なので、スペクトルで分解すると
$\sum_\beta \alpha_\beta^{j+1} e^{i\beta x_i} = \sum_\beta \alpha_\beta^j e^{i\beta x_i} + \gamma(\sum_\beta \alpha_\beta^j e^{i\beta x_{i+1}} - 2\sum_\beta \alpha_\beta^j e^{i\beta x_i} + \sum_\beta \alpha_\beta^j e^{i\beta x_{i-1}})$

任意の $\beta$ について成り立つかを考えるなら $\sum$ は外すことができて
$\alpha_\beta^{j+1} e^{i\beta x_i} = \alpha_\beta^j e^{i\beta x_i} + \gamma(\alpha_\beta^j e^{i\beta x_{i+1}} - 2\alpha_\beta^j e^{i\beta x_i} + \alpha_\beta^j e^{i\beta x_{i-1}})$
$\frac{\alpha_\beta^{j+1}}{\alpha_\beta^j} = 1 + 2\gamma(\cos\beta h - 1)$
よって、条件が成り立つためには
$\gamma < \frac{1}{2}$
が成り立てばいいですね。ちなみに $\gamma$ は正の実数という設定なので負については考えません。

陰的スキームも同様に行えば
$\frac{\alpha_\beta^{j+1}}{\alpha_\beta^j} = \frac{1}{1+2\gamma(1-\cos\beta h)}$
となり、$\gamma$ によらないことが解ります。こういうスキームは**無条件安定**と呼ばれています。今回は式の形の影響で陰的スキームに軍配が上がりましたが、必ずしも陰的スキームが優れているとは限らないので注意してくださいね。

# 7 陽的・陰的解法時間発展 ver の纏め

・関数の初期値を計算機に覚えさせ、境界条件や、偏微分方程式の近似を用いて dt 後の発展を見ていく
・近似の精度はテーラー展開で見ることが可能で、近時の方法も式に合わせて変える必要がある
・刻み幅によって誤差が発散してしまう最大値問題がある

# 8　固有値問題

（ナイフに刺されながら）
はぁ・・・はぁ・・・・・・・
こ、固有関数を求める問題も折角なので扱いましょう。
$\frac{d^2u}{dx^2} + k^2u = 0$・・・・★
境界条件は両端の値は 0 とします。これはおなじみの固有値問題ですね。答えは言うまでもなく三角関数です。

$x$ の定義領域で $u_0$ から $u_{n+1}$ までの分点を作るとすると、両端は条件から与えられているので未知数は全部で $n$ 個。また、差分法で偏微分の形を近似するとすれば、★式を近似して作ることの出来る式は全部で $n$ 個なので何とか解けそうですね。さて、四の五の言わずに行列作りましょう。★式の近似を
$\frac{d^2u_i}{dx^2} = \frac{u_{i+1}-2u_i+u_{i-1}}{h^2}$ として・・・

$\bigstar \rightarrow \frac{u_{i+1}-2u_i+u_{i-1}}{h^2} + k^2u_i = 0 \rightarrow u_{i+1} - 2u_i + u_{i-1} + \lambda^2 u_i = 0$（$\lambda^2 = h^2k^2$）
行列に直せば

$$\begin{bmatrix} 2 & -1 & 0 & \cdot\cdot & 0 \\ -1 & 2 & -1 & \cdot\cdot & 0 \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ 0 & \cdot\cdot & -1 & 2 & -1 \\ 0 & \cdot\cdot & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \cdot\cdot \\ u_{n-1} \\ u_n \end{bmatrix} = \lambda^2 \begin{bmatrix} u_1 \\ u_2 \\ \cdot\cdot \\ u_{n-1} \\ u_n \end{bmatrix}$$

見ての通り行列の固有値を求める問題に帰結します。どうやって固有値を求めるかは次の番組にお任せしまあああぁあああ

つ・か・ま・え・た・♪
さて、覚悟はいいかしら？

きゃあああああぁ（ぴちゅーん

えー。今回のランナーは志半ばでぴちゅってしまったようです・・・
それでもチャリティーは続投中なのでみなさん、じゃんじゃん投げ銭してくださいね♪
番組はまだまだ続きます。

# 9 ソースコード

しくしく、貧血でソースコードが全部は公開できないの・・・・・
一部、血でソースコードが汚れてるけど許してね（はぁと
これもみんなの宿題のためなのよ・・・・・・・

```
#include <iostream> //cin,cout を使うのに用いる
#include <math.h> //数学計算に用いる
#include <fstream> //シミュレート結果のファイル出力用
#include <sstream> //こちらも同じく
#include <string> //文字列処理用

using namespace std;

/*
 熱伝導の方程式
 lambda (d^2 f/dx^2)= df/dt を x=[0:1] 解く
 端の条件 両端で 0 を用いる
*/

int main()
{
 double getF0(double x);//関数の宣言初期値を持ってくるのに便利

 //定数
 const int xbunten=11;//x の分点の数
 double lambda=1;//らむだ
 double xrange=1;//x の定義領域 [0:xrange]
 double dt=0.00166666666;//t の格子の大きさ
 double h=xrange/(xbunten-1); //x の刻み幅
 int tstep=120;//何回先の t まで見るか

 //シミュレートする値
 double fxt[xbunten];//関数の値

 //ファイル出力用
 std::ostringstream os[xbunten];//ファイル保存用その１
 ofstream outfile("result.txt");//result.txt が自動で作られる。
```

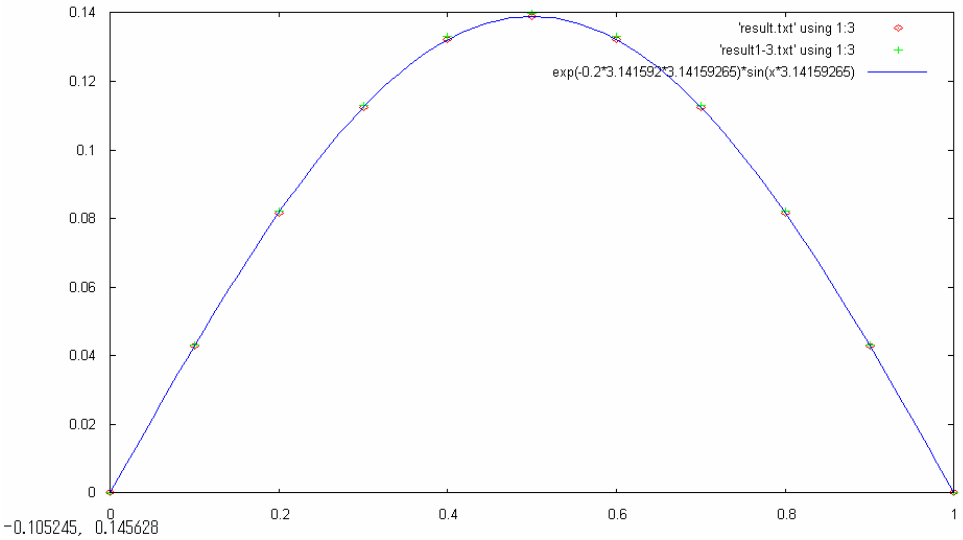
```
 //関数初期化
 for(int i=0;i<xbunten;i++){
   double s = i;//変数を double に直さないと駄目。
   fxt[i]=getF0(s*h);//i のままだと不都合が生じる可能性アリ。
   cout<<"f(0,"<<h*s<<")="<<fxt[i]<<endl;//初期値を表示
   os[i]<<h*s<<"\t"<<fxt[i];//初期値を保存。
 }

 //時間変化の導入 tstep 回行う。
 for(int i=0;i<tstep;i++){
   double temp[xbunten];//次のステップの値
   for(int j=0;j<xbunten;j++){
   double s=j;
   //端の条件
   if(j==0 || j==xbunten-1){
     temp[j]=getF0(s*h);//境界条件を用いる
   }else{
%     temp[j]=fxt[j]+dt*lambda*(fxt[j+1]-2*fxt[j]+fxt[j-1])/(h*h);
     temp[j]=【血まみれで読めません】
     //差分近似に従った変化
   }
 }
 for(int j=0;j<xbunten;j++){
   fxt[j]=【血まみれで読めません】
 }

 }
 double t=tstep;
 //結果の出力。tstep 回の時間変化後の値を出力する。
 for(int j=0;j<xbunten;j++){
  cout<<"f("<<t*dt<<","<<h*j<<")="<<fxt[j]<<endl;
  os[j]<<"\t"<<fxt[j];
 }

 //ファイルへの出力
 for(int i=0;i<xbunten;i++){
   outfile<<os[i].str()<<endl;
 }
 return 0;
}
```

```
double getF0(double x){
  return sin(3.141592 * x);
}
```

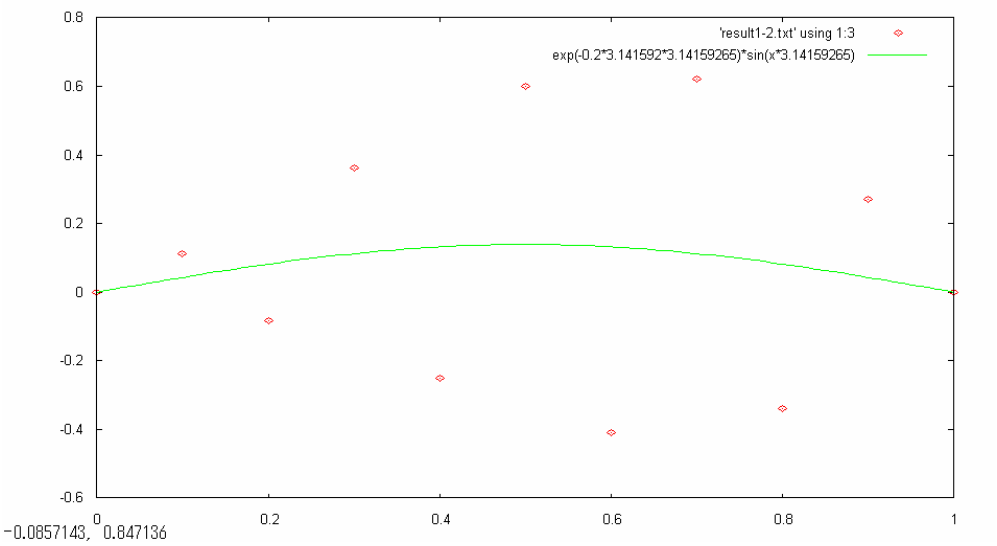
# 10 テスト結果

シミュレート結果はこんな感じになるわ。このシミュレーションで得られるのは $t = 0.2$ の時の近似解よ。理論的な解は
$exp(-0.2\pi^2)\sin(\pi x)$
ね。この章で紹介されたとおり、$\gamma$ に依存した精度で結果が出てくるわ。h ちなみに、どのシミュレートも $h = 0.1$ で行われているわ。ソースコード見れば解るわね。

無事にプログラムが完成したら色々弄ってみるといいわ。さらにカスタマイズして、cin とかで $h$ や $dt$ や $tstep$ を後から変更できるようにするとより、実験がしやすくなるわね。




出力結果 1。若干ではあるが、$\gamma = \frac{1}{6}$ の方が $\gamma = \frac{1}{10}$ より精度のよい答えとなっている。




出力結果２。$\gamma = 1$ にするとこのような悲惨な結果に。もう少し時間経過させると更に振動が激しくなる。