Lab 1 实验报告

PB20000296 郑滕飞

1、功能实现

实验要求：实现切比雪夫插值算法并验证其收敛性。

代码结构：

main.m 主程序脚本[调用 chebyshev_approx、inf_diff]

chebyshev_approx.m 计算切比雪夫插值的系数[调用 my_fft]

my_fft.m 计算快速傅里叶变换

inf_diff.m 计算插值函数与实际函数的误差，并有绘图功能

2、算法简述

对正整数 N，切比雪夫插值的表达式如下:

$$f(x) \approx \sum_{i=0}^{N} a_i T_i(x), T_i(x) = \arccos(i * \cos(x)), x \in [-1,1]$$

通过复数推导可得，记

$$y_i = f\left(cos\left(\frac{i\pi}{N}\right)\right) \quad i = 0, \dots, N$$

$$y_i = y_{2n-i} \quad i = N+1, \dots, 2N-1$$

则

$$a_i = \frac{FFT(y)}{N(1+\delta_i^0+\delta_i^N)}$$

其中δ为克罗内克符号。

于是，主函数的实现为(这里考虑了 MATLAB 中下标从 1 开始):

```
y = zeros(1, 2 * N);
y(1:N+1) = f(cos(0:pi/N:pi));
y(N+2:2*N) = flip(y(2:N));
Y = my_fft(y);
a = real(Y(1:N+1)) / N;
a(1) = a(1) / 2;
a(N+1) = a(N+1) / 2;
```

由于 y 的对称性，可发现 a 事实上是实数，因此 FFT 后可丢弃虚部。

my_fft 的算法参考自《算法导论》，先将 a 位逆序置换到 A，再进行蝶翼操作循环处理。这其中，较为麻烦的是位逆序置换，由于 MATLAB 不允许直接的移位操作，这里采用了判断二进制从低位到高位，而不断乘 2 可以把位抬高的处理思路计算 0 开始的下标 i 对应的下标 r:

```
    t = int32(i);
    r = int32(0);
    for j = 1:n
        if (mod(t, 2) > 0)
            r = r + 1;
```

```
            t = t - 1;
        end
        t = t / 2;
        r = r * 2;
    end
    r = r / 2;
```

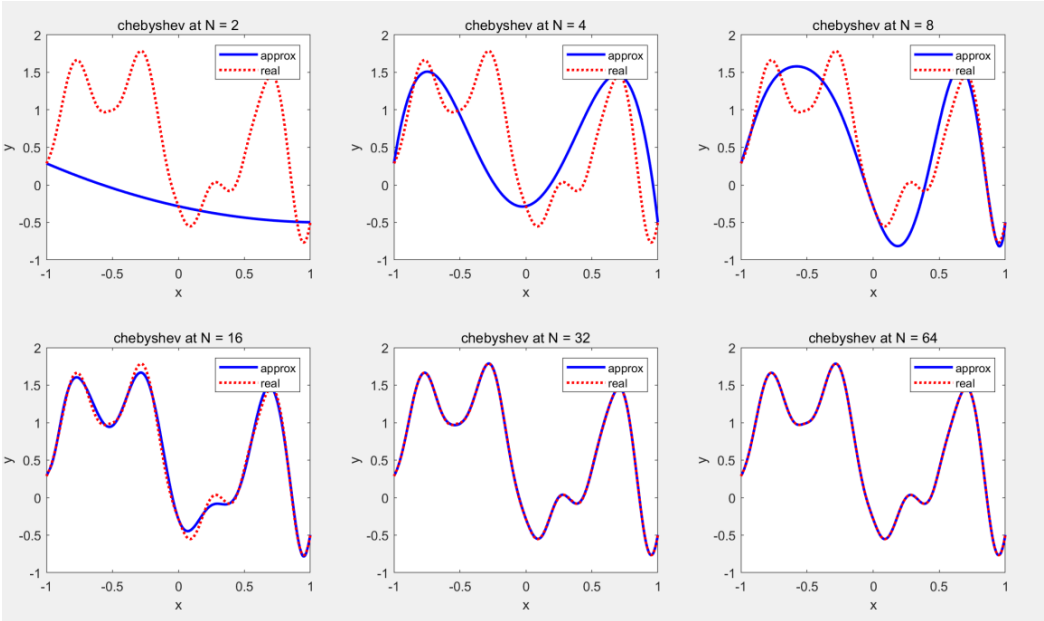
蝶翼操作为(同样考虑下标从 1 开始):

```
m = 1;
for s = 1:n
    m = m * 2;
    omega_m = exp(2 * pi * 1i / m);
    for k = 0:m:N-1
        omega = 1;
        for j = 1:m/2
            t = omega * A(k+j+m/2);
            u = A(k+j);
            A(k+j) = u + t;
            A(k+j+m/2) = u - t;
            omega = omega * omega_m;
        end
    end
end
```

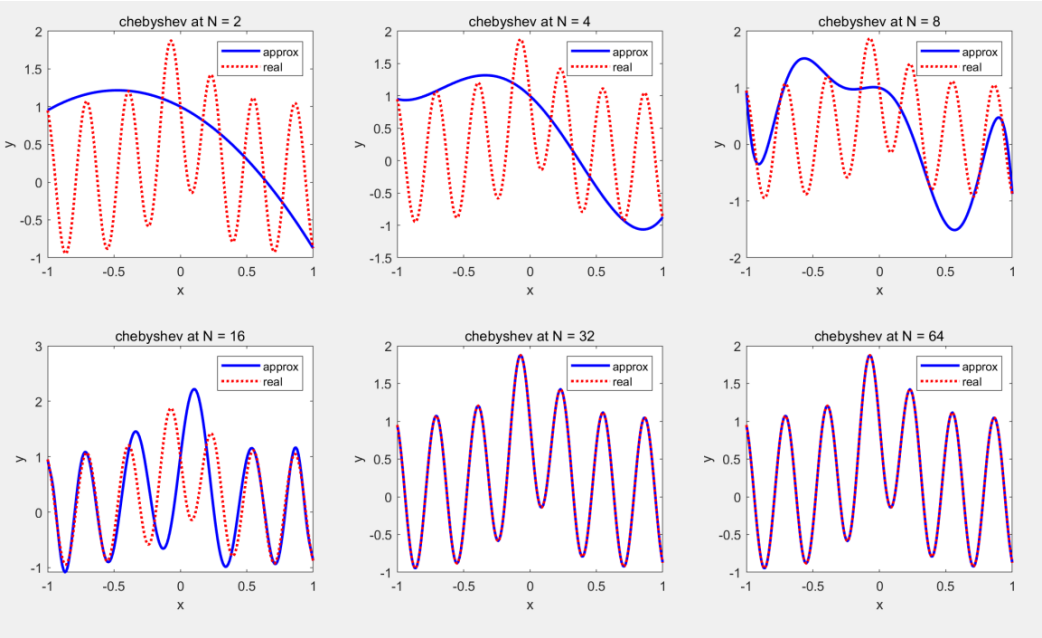
3、基本效果

以下记$f_1(x) = |\sin^3(6x)| - \cos(5e^x)$, $f_2(x) = \frac{1}{1+25x^2} - \sin(20x)$。

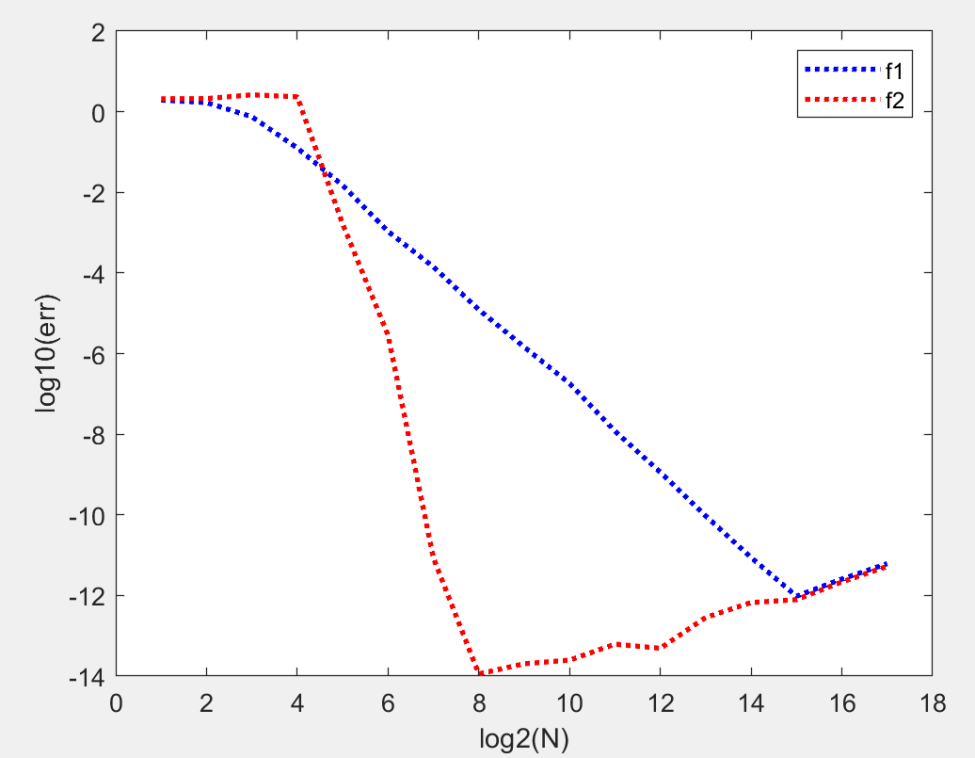
对函数 f1 的基本效果如下:



对函数 f2 基本效果如下：



可以发现，对两个函数，当 N = 32 时已经基本无法肉眼看出两函数的差别，不过，仍然能计算出误差。以下通过均匀取 400 个点处的最大误差来作为误差(也即两函数差的无穷范数)，得到固定 N 下对两函数的误差作图为：



部分误差如下：

N	err1	err2
2	1.945760e+00	2.094456e+00

4	1.685692e+00	2.113439e+00
8	7.484643e-01	2.622815e+00
16	1.256133e-01	2.340127e+00
32	1.483636e-02	1.636613e-03
64	1.069275e-03	2.857562e-06
128	1.412774e-04	8.436696e-12
256	1.224482e-05	1.132427e-14
512	1.452483e-06	2.020606e-14
1024	1.815264e-07	2.486900e-14
2048	1.191107e-08	6.195044e-14
4096	1.136536e-09	4.884981e-14
8192	9.365875e-11	2.855494e-13
16384	8.713696e-12	6.665779e-13
32768	9.580114e-13	7.858159e-13

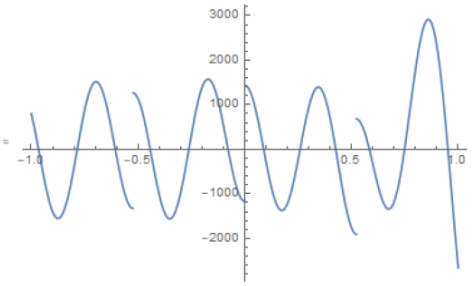
可以发现，对 f2，切比雪夫插值有着更好的收敛性，并且在 N 增大时，机器精度引起的舍入误差更加明显。而对 f1，则收敛性相对较差。从直观上，f2 比 f1 有更好的光滑性，因此结果是自然的，接下来详细考察。

4、可微函数误差界［定理 2 验证］

对光滑性不够良好的函数，其切比雪夫多项式收敛速度是 N 的多项式量级。

接下来以 f1 为例，由其包含绝对值的三次方可发现其至多有二阶连续导数(由有限点处不影响，认为绝对值导数为符号函数，符号函数导数恒 0)，于是定理 2 中取 m = 3。接下来计算三阶导数的变差。先计算化简三阶导数，得到如下函数(细节过程见 cal1.nb)：

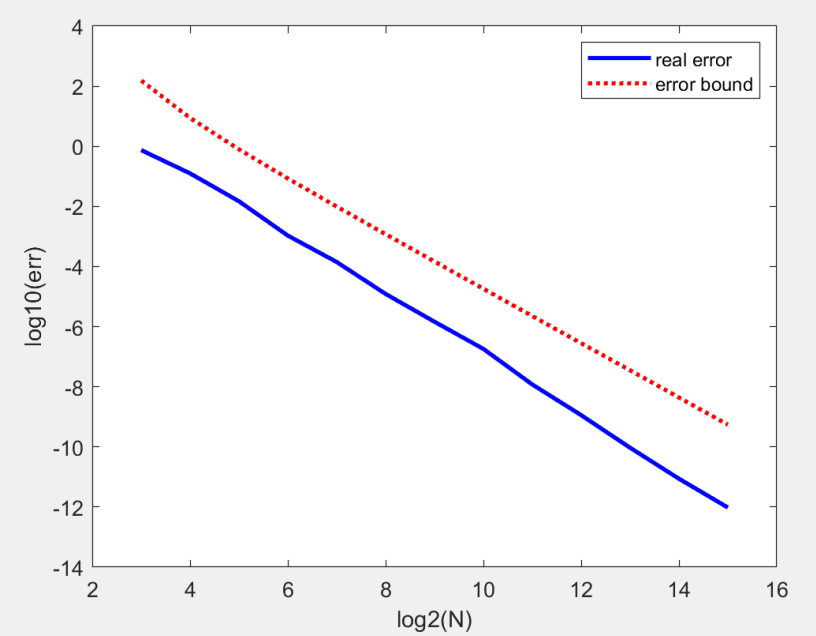
```
Plot[75 e^(2 x) Cos[5 e^x] + 1296 Cos[6 x]^3 Sign[Sin[6 x]] - 5 e^x (-1 + 25 e^(2 x)) Sin[5 e^x] - 2268 Abs[Sin[6 x]] Sin[12 x], {x, -1, 1}]
```



为计算全变差，先考虑 sin(6x)正负不发生变化的四个区域的全变差(这四个区域内分别无穷次可微，因此即为其导函数绝对值积分)，而区域交界处由于 cos(6x)的绝对值都为 1，总共三次跃变引起的全变差为 6*1296，于是最终结果：

```
In[5]:= f1 = 75 e^(2 x) Cos[5 e^x] + 1296 Cos[6 x]^3 - 5 e^x (-1 + 25 e^(2 x)) Sin[5 e^x] - 2268 Sin[6 x] Sin[12 x];
        f2 = 75 e^(2 x) Cos[5 e^x] - 1296 Cos[6 x]^3 - 5 e^x (-1 + 25 e^(2 x)) Sin[5 e^x] + 2268 Sin[6 x] Sin[12 x];

In[7]:= V = NIntegrate[Abs[D[f1, x]], {x, -1, -Pi/6}] + NIntegrate[Abs[D[f2, x]], {x, -Pi/6, 0}] + NIntegrate[Abs[D[f1, x]], {x, 0, Pi/6}] +
          NIntegrate[Abs[D[f2, x]], {x, Pi/6, 1}] + 6*1296;

In[8]:= NumberForm[V, 16]

Out[8]//NumberForm=
        45459.99166524233
```

由此，取全变差上界 45460，代入公式后得到估算整体误差结果，联合真实误差取对数作图：



可以直观看出，实际误差的收敛阶(即斜率)与估算基本一致。此外，定理 2 还包含了切比雪夫多项式系数的上界，得到的具体数据为：

N	coe dis	real err	err bound
8	4.614952e+01	7.484643e-01	1.543506e+02
16	9.771952e-01	1.256133e-01	8.781895e+00
32	3.800095e-02	1.483636e-02	7.910871e-01
64	2.083652e-03	1.069275e-03	8.500193e-02
128	1.108246e-04	1.412774e-04	9.878437e-03
256	6.083985e-06	1.224482e-05	1.191398e-03
512	3.676650e-07	1.452483e-06	1.463069e-04
1024	2.322017e-08	1.815264e-07	1.812763e-05
2048	1.319288e-09	1.191107e-08	2.255996e-06
4096	9.105880e-11	1.136536e-09	2.813799e-07
8192	4.799754e-12	9.365875e-11	3.513385e-08
16384	1.708945e-13	8.713696e-12	4.389318e-09
32768	-2.031938e-13	9.580114e-13	5.485141e-10

这里 real err 与 err bound 即为实际误差与误差上界，而 coe dis 则代表在给定 N 下的系数上界减去实际系数模长中的最小值。值得注意的是，最下面一行系数的模长最大值超过了理论上界，这也是 N 很大时机器精度限制的结果，进一步增大 N 后，无论是系数误差还是整体误差都不减反增。

于是，在不考虑机器精度限制的情况下，我们的确验证了定理 2 的成立性。

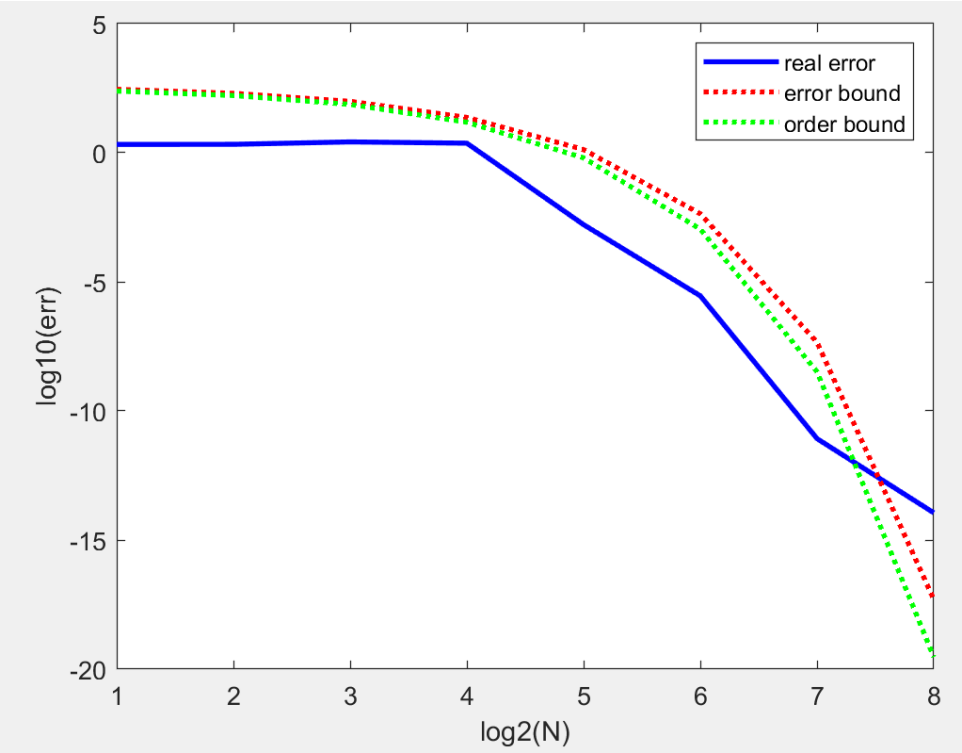
5、全纯函数误差界［定理 3 验证］

而对于可以延拓为全纯的函数，切比雪夫多项式的收敛速度就是 N 的指数量级了。

接下来以 f2 为例，由于其在复平面上只有±0.2i两个奇点，Beinstein 椭圆短轴最长为 0.2，于是可以得到理论的最优$\rho = 0.2 + \sqrt{1.04} \approx 1.2198$。不过，此时 M 会趋于无穷，因此实际上可取稍小一点的值，如$\rho = 1.19607$，并得到对应的 M 值(细节过程见 cal2.nb)：

```
In[1]:= f[x_] := 1 / (1 + 25 x^2) - Sin[20 x];
                                    正弦

In[2]:= b = 0.18;

In[3]:= a = Sqrt[b^2 + 1]
            平方根

Out[3]= 1.01607

In[4]:= NumberForm[a, 16]
        数值近似

Out[4]//NumberForm=
        1.016070863670443

In[5]:= Maximize[{Abs[f[x + y I]], x^2 / a^2 + y^2 / b^2 ≤ 1}, {x, y}]
        最大点值    绝对值      虚数单位

Out[5]= {19.9607, {x → 0.0129438, y → -0.179985}}
```

取 M = 20，代入公式后得到估算整体误差结果(红线)，联合真实误差(蓝线)取对数作图：



值得一提的是，这里的绿线是仍取 M 为 20，并取ρ为理论上界值后，得到的比原公式更紧的上界。除去 N = 8 时已经超过了机器精度的理论精度，可以发现在此例子中事实上上界是可以改进的。

联合定理 3 中切比雪夫多项式系数的上界，得到的具体数据为：

N	coe dis	real err	err bound
2	1.948077e+01	2.094456e+00	2.852106e+02
4	1.932233e+01	2.113439e+00	1.993666e+02
8	9.466584e+00	2.622815e+00	9.741505e+01
16	2.259339e+00	2.340127e+00	2.325805e+01
32	1.292921e-01	1.636613e-03	1.325769e+00
64	4.211387e-04	2.857562e-06	4.307811e-03
128	4.455239e-09	8.436696e-12	4.548146e-08
256	-2.265528e-15	1.132427e-14	5.069789e-18

这里后三列的含义与上一部分相同，coe dis 仍表示给定 N 下的系数上界减去实际系数模长中的最小值，也仍因为 N 很大时机器精度的限制导致最后一行并没有达到理论结果。而在可以保证精度的范围内，定理 3 的成立性已经得到了验证。

6、总结

从以上结果中，不难看出切比雪夫多项式逼近的优越性：它避免了结点插值容易发生的龙格现象，并且 FFT 保证了对于 N 个结点的系数计算复杂度仅有$O(N\log N)$量级，此后计算每个点的值也只有$O(N)$量级。

另一方面，哪怕对并不太好的函数，切比雪夫多项式逼近也能做到相对高阶收敛(例如 f1 的三阶收敛)，更好的函数更是能达到指数收敛，这意味着无需很大的 N 就能达到相当好的逼近效果。

在实践中可以观察得到，想要进一步精确化，需要解决机器精度问题，也即避免过程中引起的过多有效数字损失，这意味着在中间部分可能需要更多采用符号运算的方式。

附录-程序使用说明

如开头介绍，主程序为 src/main.m，其参数(均要求为非负整数)均设有默认值，因此可以直接在命令行窗口输入 main 以运行，接下来介绍参数具体含义(以下插值指切比雪夫多项式插值，无穷范数通过[-1,1]均匀取 400 个点以估计)：

basic:

非零时展示 f1、f2 的$2,4,\dots,2^{basic}$阶插值的图像与原图像的对比，并输出为两张图，默认值为 6，默认参数时效果即报告中的前两张图。

plt_err:

非零时，在控制台打印两函数$2,4,\dots,2^{plt_err}$阶插值与原函数误差的无穷范数，并将误差取对数后作图，默认值为 17，默认参数时部分控制台输出与作图即第三部分最后。

diff_err:

对应定理 2，非零时，在控制台打印 f1 的$8,16,\dots,2^{diff_err+2}$阶插值(定理 2 在 N 至少为 5 时方对系数有意义)与原函数误差的无穷范数与估算上界的对比，及系数相关对比，并将误差与上界绘图，默认值为 13，默认参数时输出与第四部分一致。

hom_err:

对应定理 3，非零时，在控制台打印 f2 的$2,4,\dots,2^{hom_err}$阶插值与原函数误差的无穷范数与估算上界的对比，及系数相关对比，并将误差与上界绘图，默认值为 8，默认参数时输出与第五部分一致。

Lab 2 实验报告

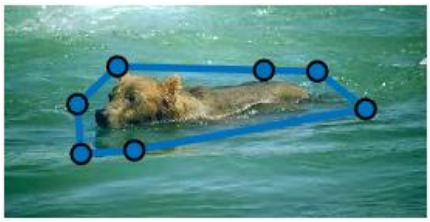
PB20000296 郑滕飞

1、功能实现

实验方向：图像融合
实验内容：通过小波变换综合其他优化方法（如泊松融合），将给定图像上的选区迁移到另一图像上给定位置。
代码结构：
main.m 主程序脚本
show.m 效果展示
poisson_editing.m 泊松融合
merge.m 系数序列融合
eql.m 均衡化
contract.m 对比度提升
figs 图片文件夹(报告中使用图片)

2、基本处理

确定图片后，可利用 MATLAB 自带的 draw_polygon 方法在图上进行交互选区：



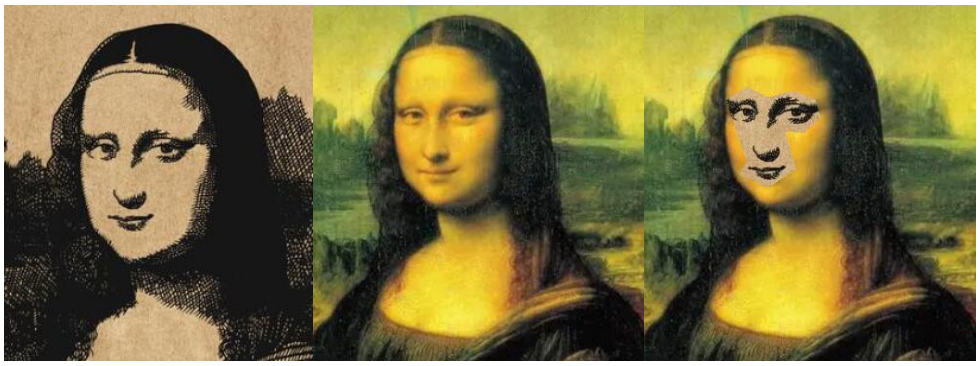
接着利用 draw_point 工具选择第二个图像中的插入点(这个点对应的是多边形第一个顶点平移到的位置，用它确定 x、y 的偏移)：

直接计算偏移后复制可以得到粘贴的效果：

以下将原图片记为 I，目标图片记为 J，粘贴结果记为 K。

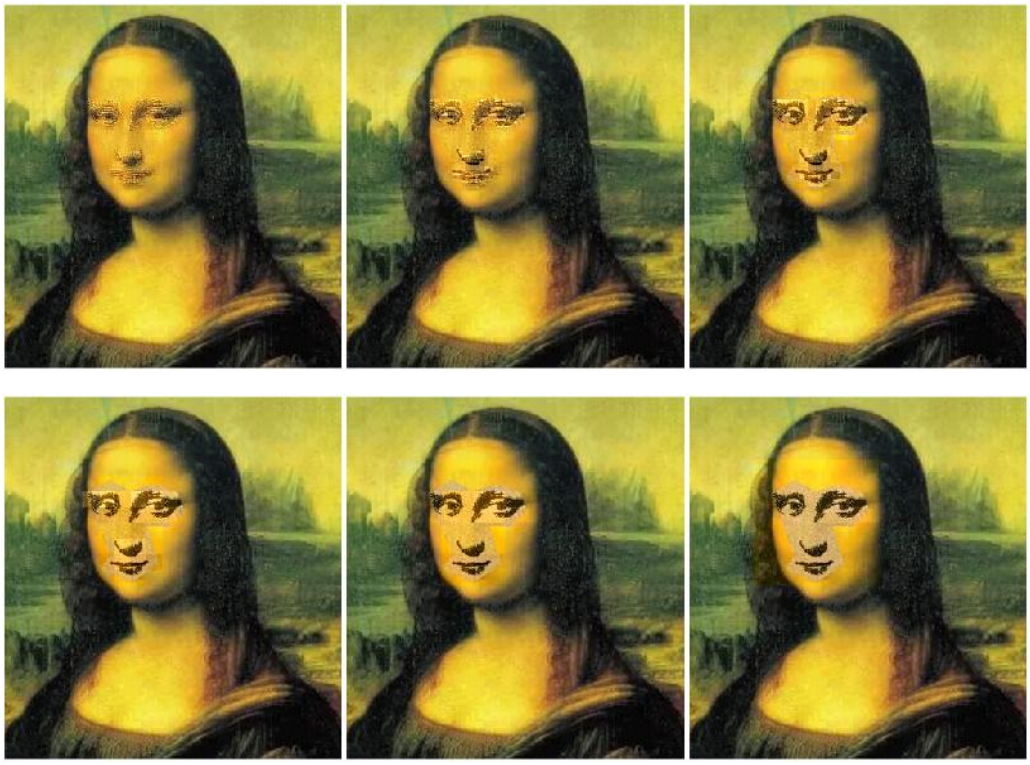
3、小波变换

为了利用小波变换达到自然融合的效果，我们考察如下的方便处理的例子，I、J、K 分别为：



由于利用小波变换进行图像处理在图像大小相同时系数才有整合的意义，我们主要考察 J 与 K 的变换。

最简单的想法即为直接替换：将 J 或 K 中的部分低频或高频系数替换为另一张的对应系数。由于“特征”主要由高频系数体现，而颜色风格等信息来自低频，尝试将某级小波分解中最低频部分的 J 的系数组合其他 K 的系数，1 到 6 级效果分别为：



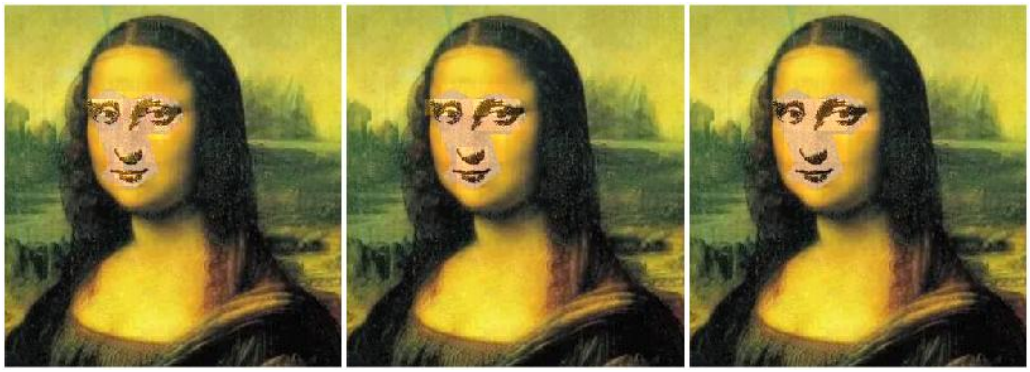
可以发现，替换低频系数后处理的效果相当于将选区划分为一些方格（变换 n 次对应方格边长为2^n），并调整每个方格内的颜色使之与原图尽量接近。但是，当变换次数太小，方格划分太精细，会直接逼近 J 而看不出 K；变换次数太大时，又导致明显的方格边界，替换

的系数不足时甚至效果逼近 K 而看不出 J。

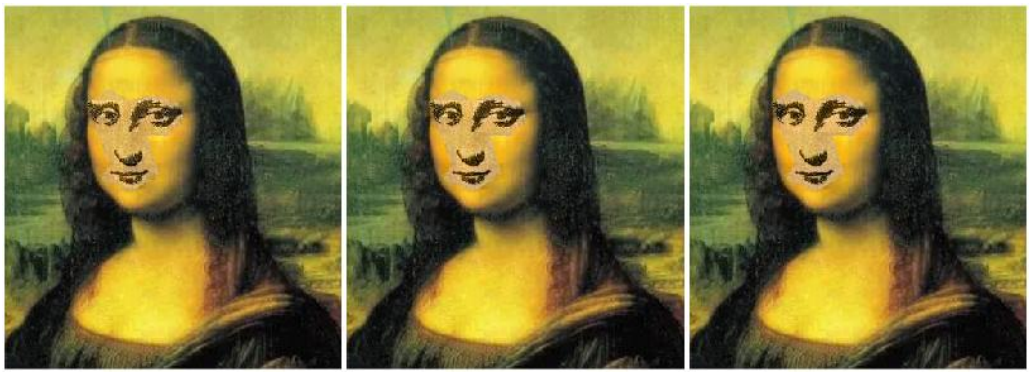
在上方的六张图里，变换 3 到 5 次是较合适的选择，基本在 K 的图像基础上贴近了 J 的颜色，下面考虑在此基础上进行进一步的优化。下一节左中右三张图一般分别代表小波分解 3、4、5 次后。

4、系数融合

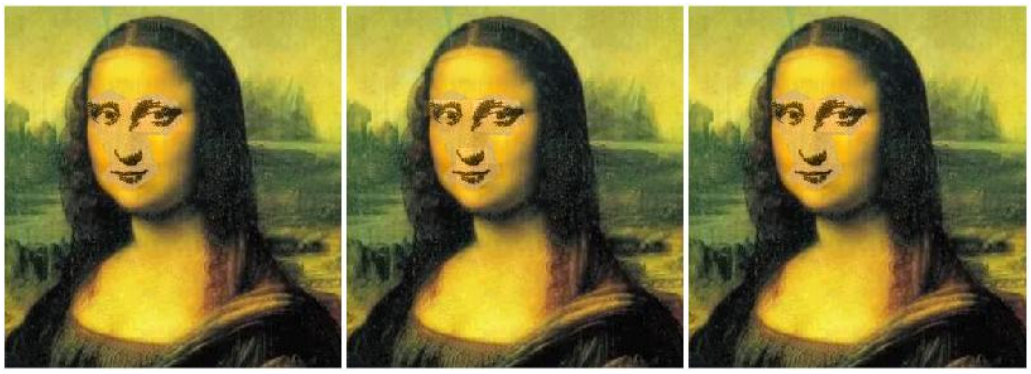
一个简单的想法是，比起直接替换，我们可以尝试不同的系数融合方式，例如取最大值（事实上应取模最大，不过由于原图为正，低频系数不会低于 0）：



或进行线性组合（如下为直接平均）：



又或者将低频高频分别进行组合（如下低频 J:K 为 4:1，高频 1:4）：



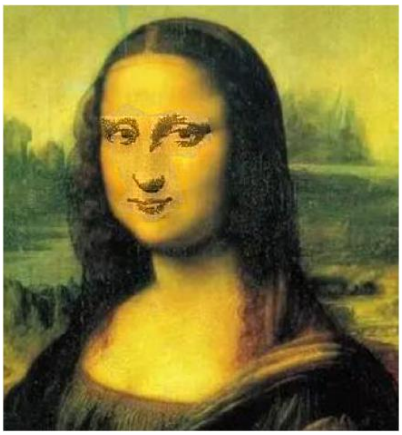
虽然无论进行怎样的直接系数融合都无法规避方格边界问题，但低频高频分别进行组合时，边界已经减弱了不少，这让我们想到对不同频率的系数进行不同比例的掺杂，越低频掺入越多的 J，越高频保留越多的 K，由此或许可以构造出更自然的结果。

于是，构造函数 merge(X, Y, m)，其中 m 为代表各阶系数中 X 占比的列表：

```
function Z = merge(X, Y, m)
    n = length(m);
    Z = zeros(size(X), "uint8");
    for i = 1:3
        [Cx, ~] = wavedec2(X(:, :, i), n - 1, 'haar');
        [Cy, S] = wavedec2(Y(:, :, i), n - 1, 'haar');
        s = S(:, 1) .* S(:, 2);
        Cx(1:s(1)) = m(1) *  Cx(1:s(1)) + (1 - m(1)) * Cy(1:s(1));
        now = s(1);
        for t = 2:n
            ed = 3 * s(t) + now;
            Cx(now + 1:ed) = ...
                m(t) * Cx(now + 1:ed) + (1 - m(t)) * Cy(now + 1:ed);
            now = ed;
        end
        Z(:, :, i) = waverec2(Cx, S, 'haar');
    end
end
```

只要设置合适的 m，理论来说可以达到光滑的融合。例如，设置 m 为 0:0.2:1（也即 0 到 1 以 0.2 为公差的 6 项等差数列）的效果为：

又或者设置 m = [0, 0.1, 0.3, 0.5, 0.8, 1]效果为：



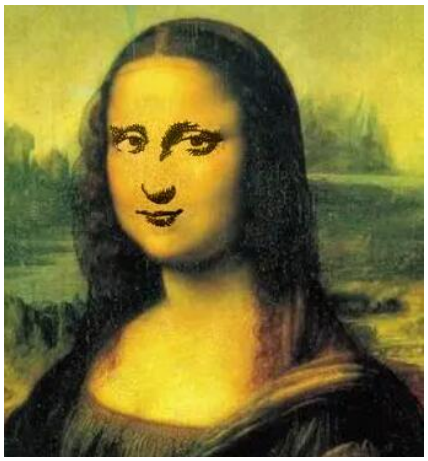
通过这样的方式，已经可以达到几乎光滑的融合。

5、效果增强

为了找到更好效果的方式，我们考察图像融合中常用算法：泊松融合。假设原图为 f，目标图片为 g，区域记为 D，泊松融合的目标为优化（梯度接近、边界一致）：

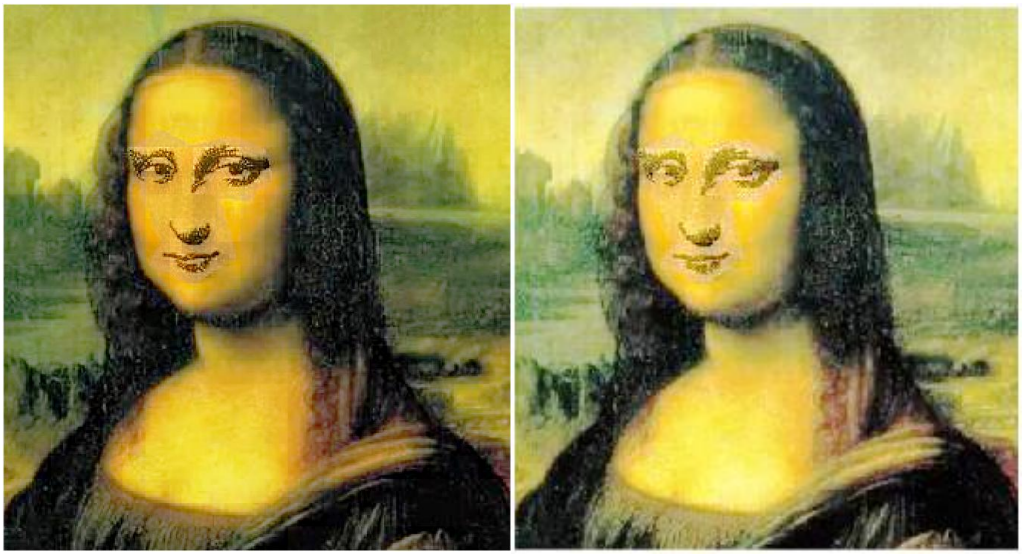
$$\min_h \iint_D |\nabla h - \nabla f|^2, \text{s.t.}\,(h-g)|_{\partial \mathrm{D}} = 0$$

并以此构造线性方程组求解。效果为：



（实际应用中，可以增加不同的梯度项与正则化项，达到混合梯度/保留部分原图的效果）

对比其与系数融合的效果可以发现，泊松融合对原图像的对比度（梯度）保留更好，因此我们希望能在直接进行融合的方法中产生更高的对比度。但是，直接对 K 通过均衡化增加对比度（左图）或对结果进行均衡化（右图）的效果并不好，尤其在图像存在主色调时：

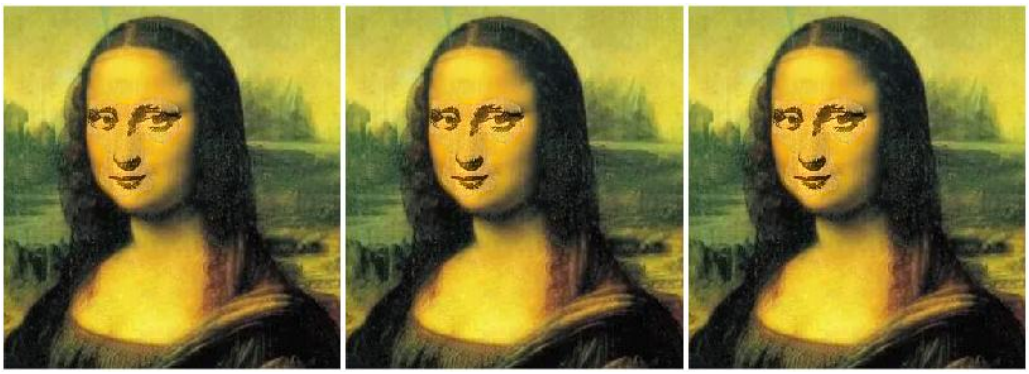


因此，我们希望在局部增强对比度：保持 region 区域内中位数颜色不变，并将其他亮度按差距映射放大。具体实现为：

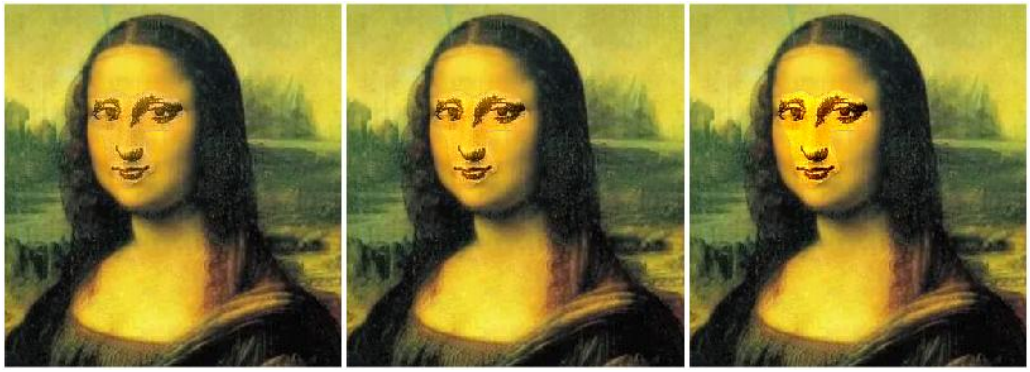
```
function J = contract(I, region, d)
    J = I;
```

```
    h = PDF(I, region);
    t = 1;
    while h(t) < 0.5 && t < 256
        t = t + 1;
    end
    [R, C, ~] = size(I);
    for i = 1:R
        for j =1:C
            if region(i, j) > 0
                J(i, j, :) = (double(I(i, j, :)) - t) * d + t;
            end
        end
    end
end

function h = PDF(I, region)
    h = zeros(256,1);
    for g = 0:255
        h(g+1,1) = sum(sum(sum(I==g, 3) .* double(region)));
    end
    h = h / sum(sum(region)) / 3;
    for g = 0:255
        h(256-g,1) = sum(h(1:256-g));
    end
end
```

先对 K 处理后得到效果（d 取 1.1、1.2、1.3）：



而直接对结果 Z 处理后则得到效果（d 取 1.1、1.2、1.3）：



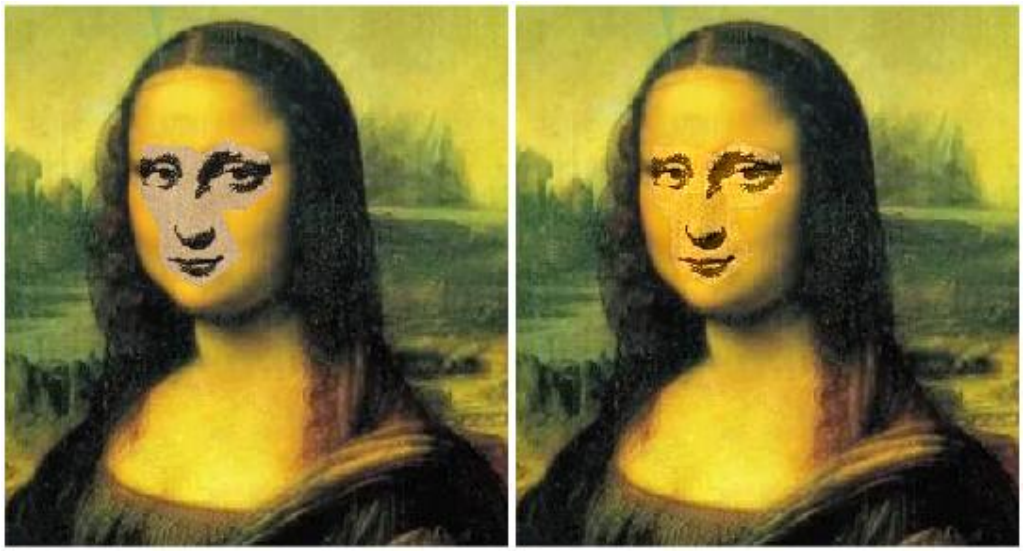
这里可以看出，本例子中对 K 进行 d = 1.2 处理的效果是较好的。值得一提的是，由于每级小波分解决定正方形大小，对不同的图像，需要取不同的融合系数序列 l，也同样需要尝试不同的 d，例如对开头熊与海的例子，最好的情况是取 l = 0:0.1:1，d = 1.3，在变换后处理，效果为（左图为直接拼接，作为对比）：

6、结果展示

在结合以上的研究后，我们最终得到了小波变换进行图像融合的较好方法，分为直接拼接、选区对比度提升（参数 d1）、序贯系数融合（参数 l）、结果选区对比度提升（参数 d2）四步。选择参数后较好的融合效果展示为：

d1 = 1.3, l = [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1], d2 = 1.3

d1 = 1.3, l = [0, 0.2, 0.4, 0.6, 0.8, 1], d2 = 1.1



d1 = 0.9, l = [0, 0.2, 0.4, 0.6, 0.8, 1], d2 = 1.1

（这里 0.9 事实上收缩了对比度）

7、总结

相较整体进行的泊松融合，小波变换的优点在于复杂度降低了很多，不需要对复杂的方程组进行求解，只需要对选区进行小波分解、系数组合与还原即可，若切割选区部分进行小波变换，复杂度基本在$O(n)$量级，n 为选中的像素数，事实上，可以在基本小波变换的基础上以泊松融合的最优性判定对系数进行梯度下降，迭代后可达到更优。与之相对，边界不规则时，泊松融合中稀疏求解的复杂度在$O(n^{2.4})$以上，因此，小波变换用于图像融合仍然有很大的实用性。

（这个 lab 最难的地方是怎么在核心部分调包的情况下整活）

Lab 3 实验报告

PB20000296 郑滕飞

实验方向：Wavelet Galerkin 方法的实现

实验内容：通过 Wavelet Galerkin 法实现对常微分方程与偏微分方程求解的程序。

一、常微分方程求解

1、基本问题

本部分我们考虑形如

$$-\big(\alpha(x)u'(x)\big)'+\beta(x)u(x)=f(x),a<x<b$$
$$u(a)=c,u(b)=d$$

的二阶常微分方程边值问题。

为方便小波求解，我们需要先进行归一化操作:

令$v(y)=u\big((b-a)y+a\big)-c-(d-c)y$，则$u(a)=c,u(b)=d\Leftrightarrow v(0)=v(1)=0$。

记$x=(b-a)y+a$，则$y_x=\frac{1}{b-a}$，记$\alpha_0(y)=\alpha(x),\beta_0(y)=\beta(x),f_0(y)=f(x)$，代入方程有

$$-(\alpha_0(y)(v(y)+c+(d-c)y)_x)_x+\beta_0(y)(v(y)+c+(d-c)y)=f_0(y)$$

计算得

$$-\frac{1}{(b-a)^2}\big(\alpha_0'(y)(v'(y)+d-c)+\alpha_0(y)v''(y)\big)+\beta_0(y)v(y)$$
$$=f_0(y)-\beta_0(y)(c+(d-c)y)$$

即

$$-\left(\frac{\alpha_0(y)}{(b-a)^2}v'(y)\right)'+\beta_0(y)v(y)=f_0(y)-\beta_0(y)(c+(d-c)y)+\frac{d-c}{(b-a)^2}\alpha_0'(y)$$

记$\alpha_1(y)=\frac{\alpha_0(y)}{(b-a)^2},f_1(y)=f_0(y)-\beta_0(y)(c+(d-c)y)+\frac{d-c}{(b-a)^2}\alpha_0'(y)$，即有

$$-\big(\alpha_1(y)v'(y)\big)'+\beta_0(y)v(y)=f_1(y),0<y<1,v(0)=v(1)=0$$

称此为归一化后的方程，之后只需对归一化后的方程考虑，系数仍记为α,β,f。

求解归一化后的方程后，还原即有

$$u(x)=v\left(\frac{x-a}{b-a}\right)+c+(d-c)\frac{x-a}{b-a}$$

具体对数组操作的代码为:

```
alpha = alpha / (b - a) ^ 2;
y = 0:(1/2^(p+m)):1;
f = f - beta .* (c + (d - c) * y) + (d - c) / (b - a) ^ 2 * alpha_d;

% v(y) = u((b - a)y + a) - c - (d - c)y
% u(x) = v((x - a)/(b - a)) + c + (d - c)(x - a)/(b - a)
res = simple_wavelet_solve(alpha, beta, f, N, p, m);
res = res + c + (d - c) * y;
```

这里 alpha_d 为 alpha 的导数，simple_wavelet_solve 处理 v(0) = v(1) = 0 的方程。

2、Daubechies 尺度空间

对任何[0, 1]上的函数w，相乘并在[0, 1]分部积分计算（注意 v 边界处为 0）可得

$$\int_0^1 \big(\alpha(y)v'(y)w'(y)+\beta(y)v(y)w(y)-f(y)w(y)\big)\mathrm{d}\,y=0$$

对有限个支集在[0, 1]中的标准正交系$\phi_0,\dots,\phi_n$，考虑v在其上的投影$\sum_i c_i\,\phi_i$，则取$w=\phi_i$必然严格满足，因此代入可知

$$A\big(\phi_i,\phi_j\big)c=b(\phi_i)$$

其中

$$A(g,h)=\int_0^1\big(\alpha(y)g'(y)h'(y)+\beta(y)g(y)h(y)\big)\mathrm{d}\,y$$

$$b(g)=\int_0^1 f(y)g(y)\mathrm{d}\,y$$

于是通过解线性方程组可得到c，从而得到v在此标准正交系下的投影。

接下来，我们取ϕ_i为 Daubechies-N 阶小波（下简称 dbN 小波）的尺度函数$\phi_{p,i}$。它们构成了尺度空间V_p的一组标准正交基。由于$V_p\subset V_{p+1}$，p 增大时近似会更加准确。

注意到，由于 dbN 小波$\phi_{p,i}$的支集为$\left[\frac{i}{2^p},\frac{i+2N-1}{2^p}\right]$，事实上有意义的下标为 0 到$2^p+1-2N$（因此须$2^p+1-2N\geq 0$），估算出它们对应的矩阵 A 与向量 b 后，即可求解 c。

3、微分与积分的估算

在解决了整体算法的思路后，还有两个细节需要处理：由于 Daubechies 小波基本没有解析表达，关于其的求导与积分（以及$\alpha'(y)$）都需要用数值的方式估计。

MATLAB 提供的 wavefun 函数可以得到[0, 2N-1]中任意充分大均匀分割的端点处$\phi(x)$表达，而假定α输入解析表达式，也可得到类似的表达，因此，数值微分问题变为从均匀分割点处的值（由支集性质可以进行零延拓）进行导数的估算。

此处直接采用泰勒级数得到的具有四阶误差的公式进行近似：

$$f'(x)=\frac{-f(x+2h)+8f(x+h)-8f(x-h)+f(x-2h)}{12h}+O(h^4)$$

对于积分，则采用复化辛普森公式进行计算：

$$\int_0^1 f(x)\mathrm{d}\,x=\frac{h}{3}\left(f(0)+2\sum_{i=2}^{n/2}f\big((2i-2)h\big)+4\sum_{i=1}^{n/2}f\big((2i-1)h\big)+f(1)\right)+O(h^4)$$

对区间[a, b]，假设给出了均分点的 y，具体积分、微分的估算代码为（这里均采用向量化处理提升运算速度）：

```
function inte = estimate_inte_4(y, a, b)
    h = (b - a) / (length(y) - 1);
    inte = (y(1) + 2 * sum(y(3:2:end-1)) ...
        + 4 * sum(y(2:2:end-1)) + y(end)) * h / 3;
end

function der = estimate_deriv_4(y, a, b)
    h = (b - a) / (length(y) - 1);
```

```
    y_e = [0, 0, y, 0, 0];
    der = ( -y_e(5:end) + 8 * y_e(4:end-1) - 8 * y_e(2:end-3) ...
        + y_e(1:end-4) ) / (12 * h);
end
```

值得注意的是，为了得到四阶的误差，需要小波至少有四阶光滑性，也即 db5 开始才能有较好的效果。若采用二阶误差的$f'(x)=\frac{f(x+h)-f(x-h)}{2h}+O(h^2)$与复化梯形积分，则只需要 db3 开始，此后会进行对比。此外，由于α端点处取值未必为 0，需要额外在端点补充点来保证估算的准确性，此外还可取更细分的点进行准确估算。对α,α'估算的代码如下，这里输入的 alpha 是函数句柄，离散精度 p + m 的含义将在下一部分解释，处理后的 alpha 与 alpha_d 即为函数与导数在2^{-m}为步长的区间剖分处的值：

```
        h_e = (b - a) / 2 ^ (p + m + 2);
        x_e = (a-2*h_e):h_e:(b+2*h_e);
        x = a:(h_e*4):b;
        fun = est;
        alpha_d = alpha(x_e);
        alpha_d = fun.deriv_4(alpha_d, a - 2*h_e, b + 2*h_e);
        alpha_d = alpha_d(3:4:end-2);
        alpha = alpha(x);
```

此外，这里矩阵 A 事实上是一个稀疏对称阵，因为只有$|i-j|<2N-1$的 i 与 j 才能使乘积与导数乘积积分非零，且有

$$\phi_{p,i}(y)\phi_{p,j}(y)=2^p\phi(2^px-i)\phi(2^px-j)$$
$$\phi_{p,i}'(y)\phi_{p,j}'(y)=2^{3p}\phi'(2^px-i)\phi'(2^px-j)$$

因此只要有ϕ与ϕ'在各点的近似，就能得到矩阵 A 与向量 b 的估算。

4、尺度函数的离散

在 MATLAB 中，wavefun("dbN", m)在 N > 1 时是将[0, 2N-1]按照2^{-m}的步长划分区间（当 N = 1 时会在右端额外给出一个区间，由于我们所有步骤都要求 N 至少为 3，无需考虑这种情况的处理）。由于估计所有$v\left(\frac{i}{2^p}\right)$处的值事实上只需要$\phi(x)$在整点处的结果，理论来说直接取 m = 0 即可得到一定程度的近似。但是，为了对导数、积分进行精确的估计，事实上应取更大的 m（由于复化辛普森积分公式要求划分区间数为偶数，m 应至少为 1）。而在实际计算中，这相当于将 y 在[0, 1]中每个$\frac{1}{2^p}$长的小区间又进一步分为了2^m段，于是事实上可以算出所有$v\left(\frac{i}{2^{p+m}}\right)$处的值，对α的离散也是由此得到。

于是，在 p 与 m 和一定时，我们得到的最小区间长度是相同的，而 p 与 m 一个作为尺度空间的维度标准，一个作为数值离散的精度标准，共同决定了结果的精度。

接下来考虑 A 与 b 的构造过程。根据之前的分析，dbN 小波 p 阶情况对应的矩阵维数应为2^p-2N+2。对于 A，由于对称性，我们事实上只需要考虑下标满足关系$i\le j<i+2N-1$的情况的计算，其他情况可直接得结果（由于 MATLAB 矩阵下标从 1 开始，为了满足 i 对应$2^{p/2}\phi(2^px-i)$，需要先将 i、j 减 1，此后默认下标从 0 开始）。由于$A(\phi_i,\phi_j)=$

$$\int_{j/2^p}^{(i+2N-1)/2^p}\left(2^{3p}\alpha(x)\phi'(2^px-i)\phi'(2^px-j)+2^p\beta(x)\phi(2^px-i)\phi(2^px-j)\right)\mathrm{d}x$$

可根据下标找到对应段点乘后相加得到积分的估计值。$b(\phi_i) = \int_{i/2^p}^{(i+2N-1)/2^p} 2^{p/2} f(x)\phi(2^p - i)\mathrm{d}x$

操作类似。在如前离散之后，代码为：

```
function a = construct_A(alpha, beta, phi, phi_d, i, j, L, p, m)
    assert(j >= i && j < i + L);
    i = i - 1;
    j = j - 1;
    fun = est;
    count = 2^(3*p) * alpha((j * 2^m + 1):((i + L) * 2^m + 1)) ...
        .* phi_d(((j - i) * 2^m + 1) : end) ...
        .* phi_d(1 : (i - j + L) * 2^m + 1);
    count = count + 2^p * beta((j * 2^m + 1):((i + L) * 2^m + 1)) ...
        .* phi(((j - i) * 2^m + 1) : end) ...
        .* phi(1 : (i - j + L) * 2^m + 1);
    a = fun.inte_4(count, j/2^p, (i+L)/2^p);
end

function b = construct_b(f, phi, i, L, p, m)
    i = i - 1;
    fun = est;
    count = 2^(p/2) * f((i * 2^m + 1):((i + L) * 2^m + 1)) .* phi;
    b = fun.inte_4(count, i/2^p, (i+L)/2^p);
end
```

代码中 `L = 2N - 1`。

而若已知系数 c，构造结果的过程则类似计算 b 时：

```
    c = A \ b;
    res = zeros(1, 2^(p+m) + 1);
    for i = 0:D-1
        res((i * 2^m + 1):((i + L) * 2^m + 1)) ...
            = res((i * 2^m + 1):((i + L) * 2^m + 1)) ...
            + phi * c(i + 1) * 2^(p/2);
    end
```

不过，由于大数浮点表示的精度损失较大，而且 p 增大时可能溢出，我们希望能一定程度减少其中的大数。注意到，所有$\phi_{p,j}$同乘一个数事实上不会影响这个算法的结果（相当于 A 乘以了其平方，而 b 与结果构造时同乘，恰好抵消），因此可以在构造 A 过程中改为

$$\int_{j/2^p}^{(i+2N-1)/2^p} \left(2^{2p}\alpha(x)\phi'(2^p x - i)\phi'(2^p x - j) + \beta(x)\phi(2^p x - i)\phi(2^p x - j)\right)\mathrm{d}\,x$$

而计算 b 与结果时都不乘$2^{p/2}$项，仍能得到正确结果。这样，最大的幂次项为2^{2p}，达到了一定的保持精度效果。

除此之外，在 MATLAB 中用 `sparse(A)`进行稀疏化后，可以有效增加这种情况的计算速度。

5、结果展示

对于书上的例子，方程为

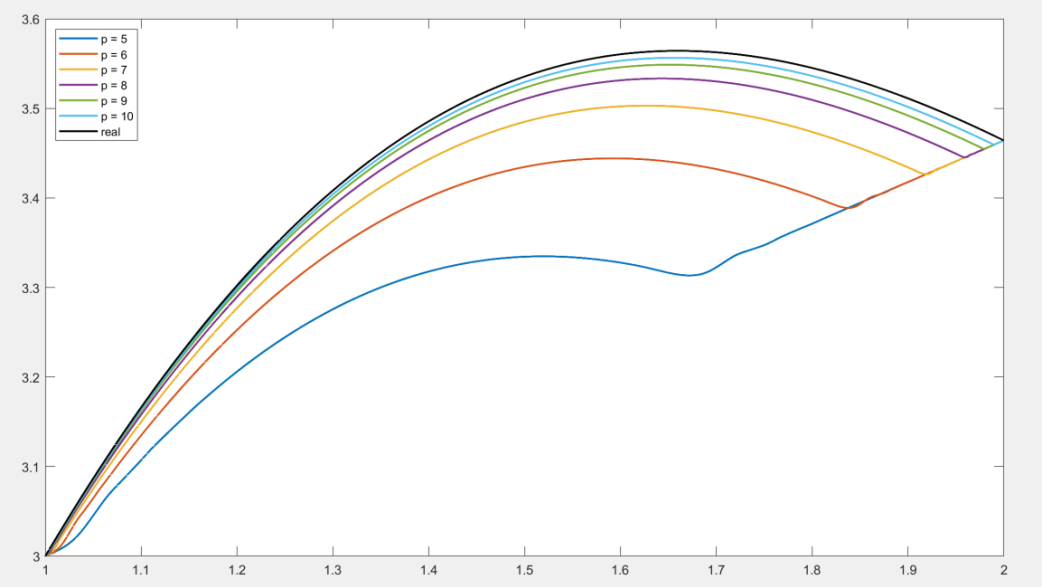
$$-u''(x)+u(x)=\left(\frac{\pi^2}{9}-4\right)\sin\frac{\pi x}{6}+\left(\frac{\pi^2}{4}-1\right)\sin\frac{\pi x}{2}$$

$$u(1)=3,u(2)=2\sqrt{3}$$

精确解为

$$u(x)=4\sin\frac{\pi x}{6}+\sin\frac{\pi x}{2}$$

取小波 `N = 7`，精度 `m = 10`，在不同的 p 进行计算，并与真实结果对比：

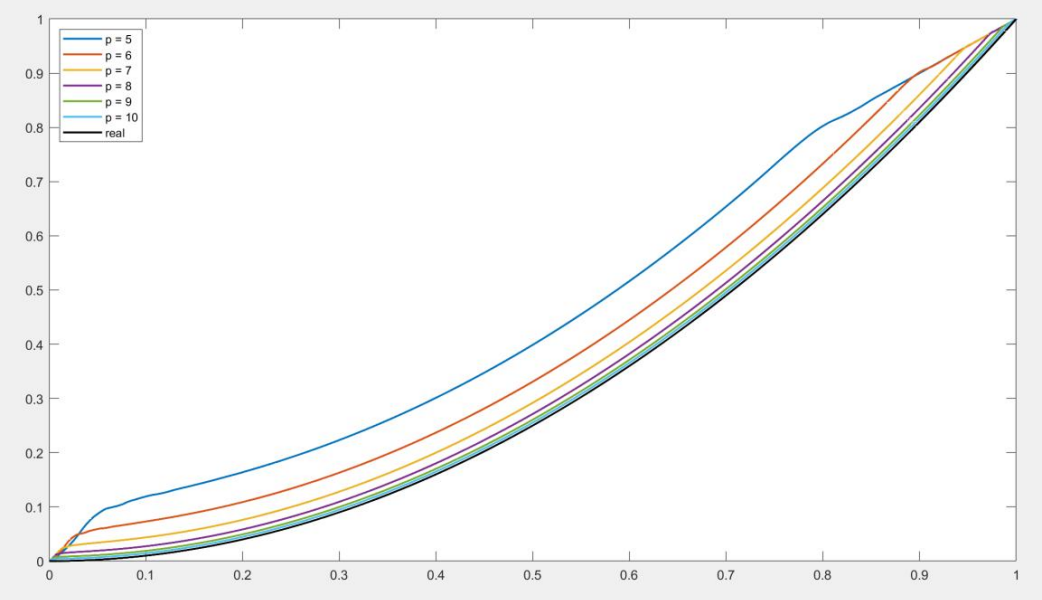


考虑另一个例子：

$$-\left(x^2u'(x)\right)'+xu(x)=x^3-6x^2$$

$$u(0)=0,u(1)=1$$

其精确解为$u(x)=x^2$，采用小波 `N = 5`，仍取 `m = 10`，对不同的 p 求解作图：



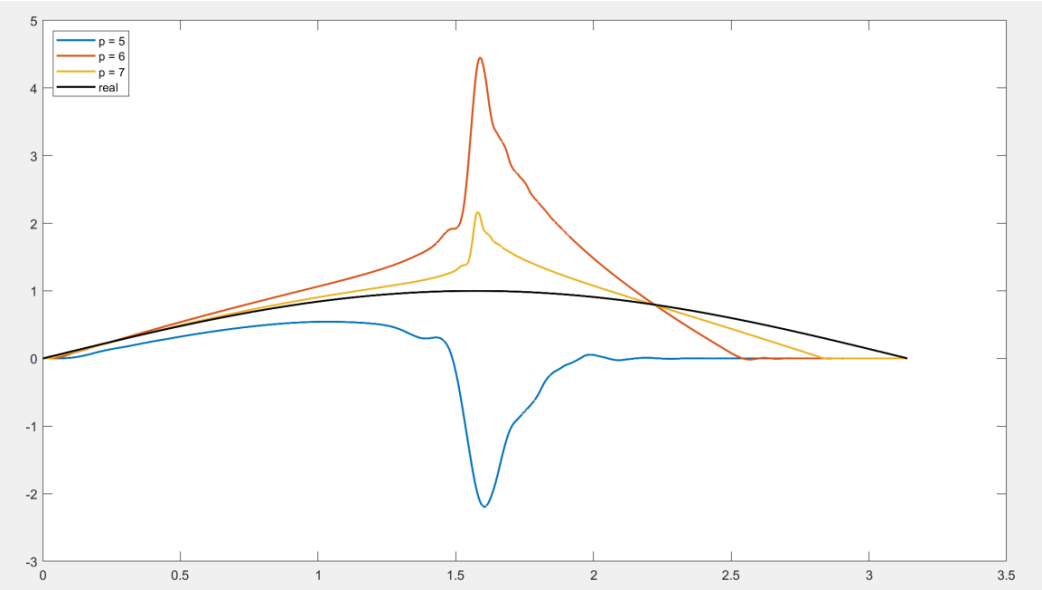
注意到，对每个解，尾部都有一部分几乎是线性插值的情况。这是由于小波在尾部消失很快，而2^p-2N+2个下标中的最后部分会存在一定长度的消失段落，近似为 0，这也就导致第一个例子中这段近似与$u=(2\sqrt{3}-3)(x-1)+3$相同，第二个例子中则与x接近 。事实上，消失段落的长度基本与最后一个支集一致并略小，也就是约为$\frac{2N-1}{2^p}$。

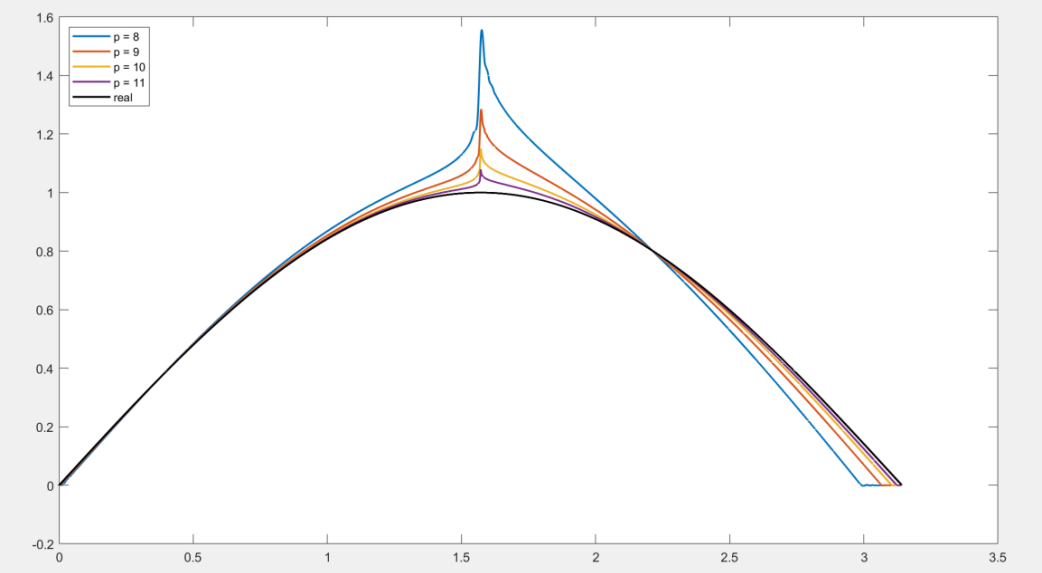
由于这段的存在，采用无穷范数考察误差对整体的刻画不够完善，于是此后对 N、p、m 进行误差分析时一律采用 L2 范数进行，也即所有估算点的误差平方的平均值开根。

最后考察一个并非单调的情况：

$$-(\cos x\,u'(x))'+\sin x\,u(x)=\sin(2x)+\sin^2 x$$
$$u(0)=u(\pi)=0$$

精确解为$u(x)=\sin x$。取 `N = 6`，`m = 10`，对不同 p 绘图：



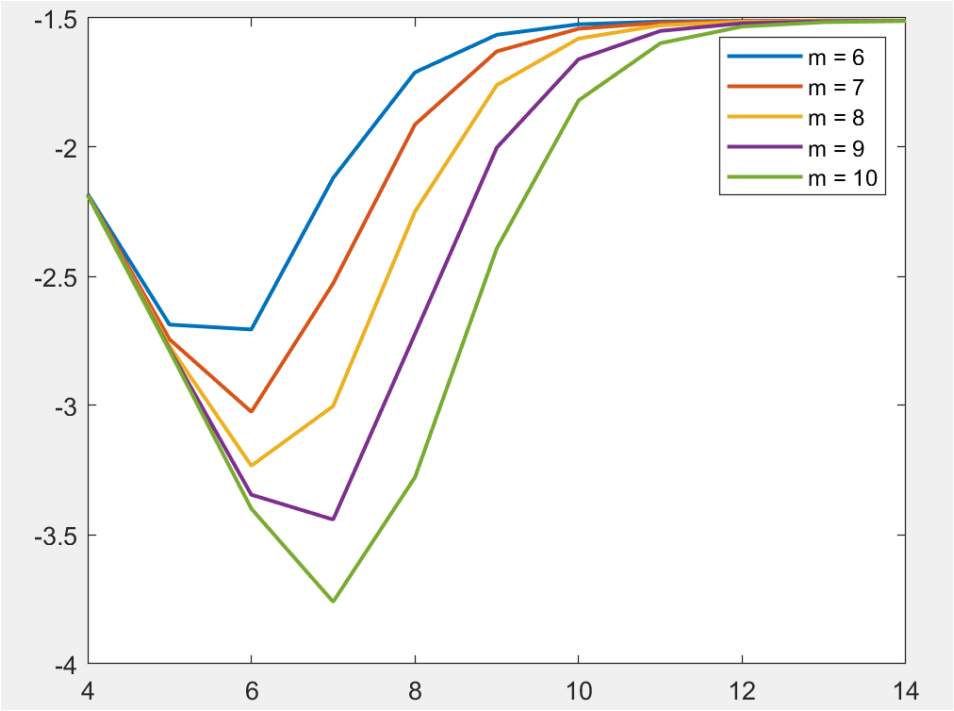


可以发现，在不单调的情况下，采用空间阶数低时可能会导致非常大的误差，而误差最大的位置也不再和最后的线性插值有关，而是发生在单调性发生改变处。不过，由于之前两

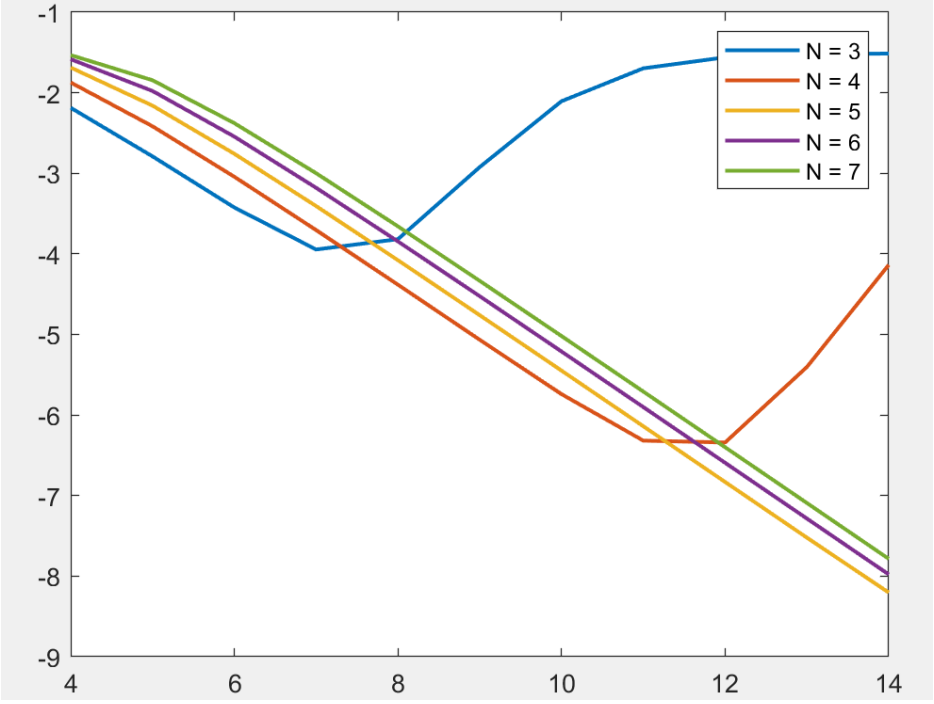
个例子在减去线性插值后也并不单调，这并不能直接说明我们的方法对单调函数效果更好，而是和函数形状的具体变化密切相关。

6、精度分析

首先观察 m 对结果的影响。在第一个例子中，取定 N = 3，以 p 为横坐标，误差对数为纵坐标绘制不同 m 时的误差图：



接下来，取定 m = 11，作不同 N 与 p 的误差图：



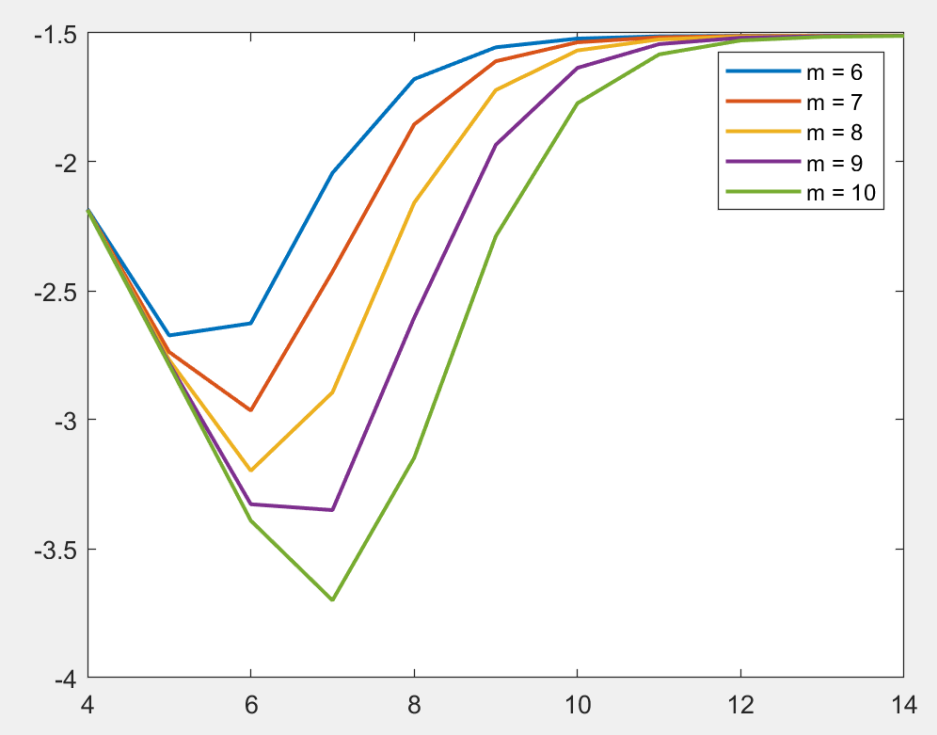
可以发现，m 增大时收敛速度加快，但总体来说对收敛阶的影响并不大。与之相对，当 m 减小时，随着 p 增大，很容易由于数值误差过大而过拟合，因此一般应取定一个较大的 m，在下文默认取定 m = 11。

而固定 m 时所得到的误差图则颇有意思。从图中可以看出，固定 m 后，随着 N 的增大，对相同的 p，误差事实上是更大的，但随着 p 的增大，N 增大时表现出了更好的数值稳定性，从而可以达到最终更低的误差。计算阶可以发现，对所有 N，阶基本恒定为 1，这个结果与书上结果的差别较大，主要原因在于归一化选择与边界处理，接下来进行分析：

首先，假设一段区间内有k个基函数，数值稳定时结果的误差可以写成$ak^{-\delta}$，$\delta > 0$的形式。我的结果中，当 N，p 取定时，误差正比于$(2^p - 2N + 1)^{-\delta}$，对同一个 p，小波的维数增大时导致了基函数个数的减小，因此误差反而上升，而当 p 充分大时，p 增加 1 导致基函数个数倍增，因此误差阶稳定于$\log_2\left(a2^{-k\delta}\right) - \log_2\left(a2^{-(k+1)\delta}\right) = \delta$，由我的结果可以得到$\delta = 1$。

而书上的情况，提到对固定的 N，p 有$(2N-1)2^p + 2N - 3$个基函数，这意味着书上算法与我的构造有两方面不同：首先，对于给定的 N 将原方程归一化到[0, 2N-1]区域，而不是[0, 1]；其次，把所有与支集与[0, 2N-1]相交的基函数（而非保持边界的严格性，只考虑支集在其中的基函数）都纳入了考虑范围内。在这样的情况下，书上的结果不会出现边界处放大的误差，且内部情况也更加精细，收缩后出现更大的收敛阶是正常的。

另一个重要的观察是数值微分/积分方式对结果的影响。如果将 4 阶误差的数值微积分公式改为 2 阶，取定 N = 3，以 p 为横坐标，误差对数为纵坐标重新绘制不同 m 时的误差图：



虽然从数值上可以计算出一定的差别，但并不明显，这也就意味着，在此情况中，主要误差来源于函数空间的选取，而非操作中的数值估计误差。

7、结果改进

根据上述的分析，虽然保证边界严格性后不能向外延伸，但将区间扩大为[0, L]可以

有效改进结果（L = 2N - 1）。虽然会导致 p 一定时矩阵计算开销更大，但可以更充分利用同一阶的 phi。在第一部分的推导中，这意味着有关系式

$$y = \frac{d-c}{L}x + c, y_x = \frac{L}{d-c}$$

于是令

$$v(y) = u\left(\frac{b-a}{L}y + a\right) - c - \frac{d-c}{L}y$$

记$\alpha_0(y) = \alpha(x), \beta_0(y) = \beta(x), f_0(y) = f(x)$，有

$$-\left(\frac{L^2\alpha_0(y)}{(b-a)^2}v'(y)\right)' + \beta_0(y)v(y) = f_0(y) - \beta_0(y)\left(c + \frac{d-c}{L}y\right) + \frac{L(d-c)}{(b-a)^2}\alpha_0'(y)$$

并且，这意味着要对α, β, f进行$L2^m$阶离散。对应操作过后，即可得到整体的结果。（值得注意的是，在进行了这样的处理后，哪怕取 p = 0 也可以存在结果）。

表达式$A(\phi_i, \phi_j) =$

$$\int_{j/2^p}^{(i+2N-1)/2^p} \left(2^{3p}\alpha(x)\phi'(2^p x - i)\phi'(2^p x - j) + 2^p\beta(x)\phi(2^p x - i)\phi(2^p x - j)\right)\mathrm{d}x$$

与

$$b(\phi_i) = \int_{i/2^p}^{(i+2N-1)/2^p} 2^{p/2} f(x)\phi(2^p - i)\mathrm{d}x$$

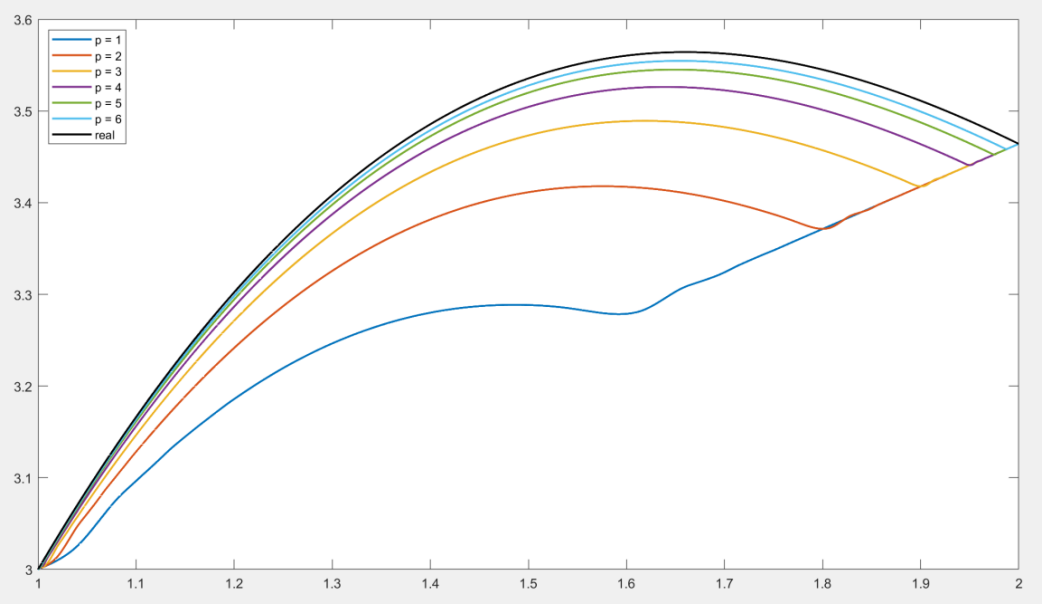
不会变化，只是 i, j 的延伸范围更大，这时基的总个数为$L2^p - L + 1$，而所有 constuct 的过程不变，在求解后，结果也取$2^{p+m}L$个网格离散，并且对所有基进行累加，还原即得最终近似解。

仍对第一个例子的方程

$$-u''(x) + u(x) = \left(\frac{\pi^2}{9} - 4\right)\sin\frac{\pi x}{6} + \left(\frac{\pi^2}{4} - 1\right)\sin\frac{\pi x}{2}$$

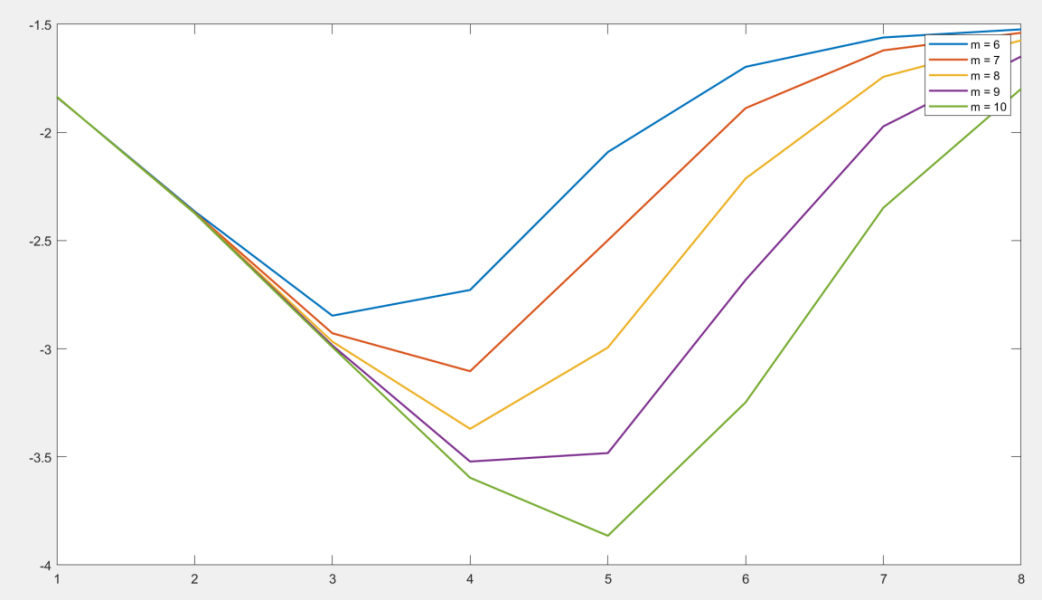
$$u(1) = 3, u(2) = 2\sqrt{3}$$

操作，从图中可以看出，在 p = 6 时就能超过之前 p = 9 的逼近程度。

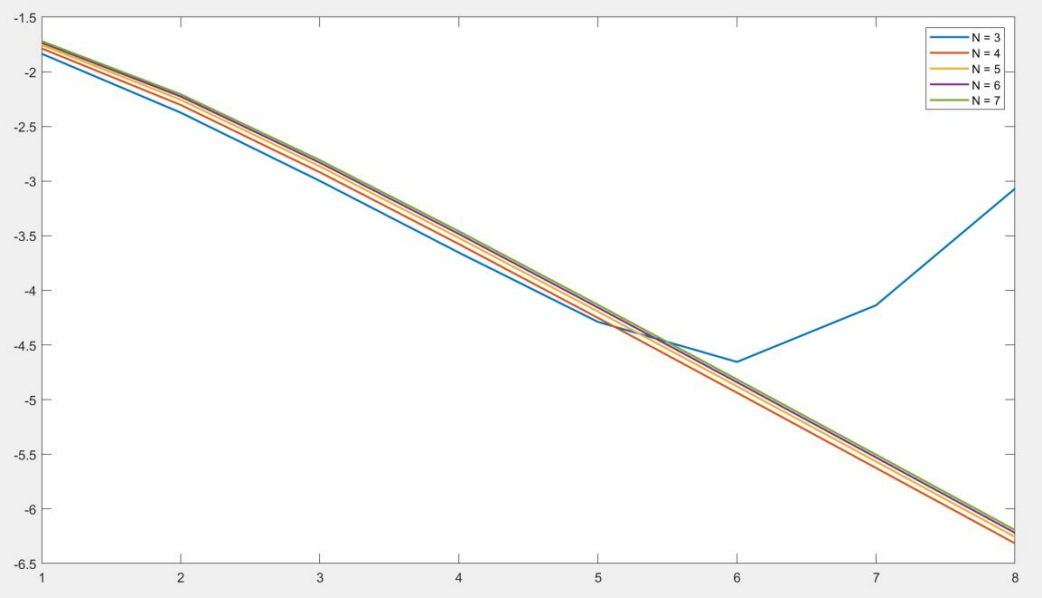


继续观察对误差的效果，下图是取定 N = 3 的情况。可以发现，数值不稳定性此时更

加明显，因此需要取到更大的 m，此后令 m = 12。



这时，考虑不同的 p 与 N 并作图：



注意下标为 1 到 8，可以明显发现此时的收敛效果好于之前。但是，仍然有 N 越大时同样 p 的结果反而越差，这可能与对导数、积分估计的数值损失与过大的 p、m 引起的步长过小相关。

二、偏微分方程求解

[部分想法参考自 www.docin.com/p-367539107.html]

1、初边值问题

在利用小波函数构造了常微分方程数值解后，我们来考察如下的偏微分方程初边值问题：

$$-(\alpha(x)u_x)_x + \beta(x)u_t = \mathrm{f}(x,t), \mathrm{a} < \mathrm{x} < \mathrm{b}, \mathrm{t} > 0$$

$$\mathrm{u}(x,0)=u_0(x),\mathrm{u}(a,t)=\mathrm{c},\mathrm{u}(b,t)=\mathrm{d}$$

类似进行归一化后，可不妨假设边界条件为$u(0)=u(L)=0$。

第一个方程两边同乘$v(x)$后对x积分可得到

$$\int_0^1(\alpha u_x v_x+\beta u_t v-fv)\mathrm{d}\,\mathrm{x}=0$$

假设$u=\sum_i c_i(t)\phi_i(x)$，则代入所有$v=\phi_i$可知

$$A(\phi_i,\phi_j)c(t)+M(\phi_i,\phi_j)c'(t)=b(\phi_i,t)$$

这里$c(t)=\left(c_i(t)\right)^T$，且

$$A(g,h)=\int_0^L\alpha(y)g'(y)h'(y)\mathrm{d}\,y$$

$$M(g,h)=\int_0^L\beta(y)g(y)h(y)\mathrm{d}\,y$$

$$b(g,t)=\int_0^L f(y,t)g(y)\mathrm{d}\,y$$

于是，结果化为了一个一阶线性常微分方程问题，初值c_0可直接通过u_0与各ϕ_i内积得到。，在利用迭代法求解$c(t)$后，即能计算出结果的表达。更进一步地，此方程可以化简为

$$c'(t)=-M^{-1}Ac(t)+M^{-1}b(t)$$

因此只需保存$M^{-1}A$与M即可计算迭代，

2、代码实现

首先写出过程的伪代码:

1. 初始化用到的各个量
2. 将 u 变换为支集在[0, L]上的 v
3. 计算矩阵 A，M，并以此得到$M^{-1}A$与M^{-1}
4. 由 v 内积计算初始的 c0

[此处可删去保存的 alpha 与 phi 导数等以节约内存]

--- Loop ---

5. 计算当前的$b(t)$
6. 根据当前的 b、c 迭代得到下一步的 c，t 自增
7. 根据 c 计算出当前的 v
8. 根据 v 还原当前的 u（可保存或输出）

--- End Loop ---

在第二步中，有

$$v_0(y)=u_0\left(\frac{b-a}{L}y+a\right)-c-\frac{d-c}{L}y$$

而后续对 f 的处理变为，预先计算好 f 的调整项 df:

```
df = - beta .* (c + (d - c) * y / L) + ...
    L * (d - c) / (b - a) ^ 2 * alpha_d;
```

接着在每次迭代中直接使用 f 计算 B:

```
fnow = f(x, t) + df;
for i = 0:D-1
    count = fnow((i * 2^m + 1):((i + L) * 2^m + 1)) .* phi;
    B(i + 1) = fun.inte_4(count, i/2^p, (i+L)/2^p) * 2^(p/2);
```

```
end
```

对常微分方程，采用最直接的欧拉方法计算 C 的迭代：

```
C = C + ht * (-T * C + M \ B);
```

这里 ht 表示时间步长。

考虑到这里可能采用不同的线性迭代格式，以及精度主要误差在于时间步，在 PDE 代码的编写中没有使用同除来增加精度。从 c 还原结果时，我将结果保存在了 gif 图片之中，用于展现不同时间 x 的变化：

```
%% count v

res = zeros(1, 2^(p+m) * L + 1);
for i = 0:D-1
    res((i * 2^m + 1):((i + L) * 2^m + 1)) ...
        = res((i * 2^m + 1):((i + L) * 2^m + 1)) ...
        + phi * C(i + 1);
end
res = res * 2^(p/2);

%% count u

res = res + c + (d - c) / L * y;

% draw and save
plot(x, res, "linewidth", 1.5);
[A, map] = rgb2ind(frame2im(getframe(fig)), 256);
imwrite(A, map, "res.gif", 'WriteMode', 'append', ...
    'DelayTime', 0.03);
axis([a, b, 0, 1]);
```

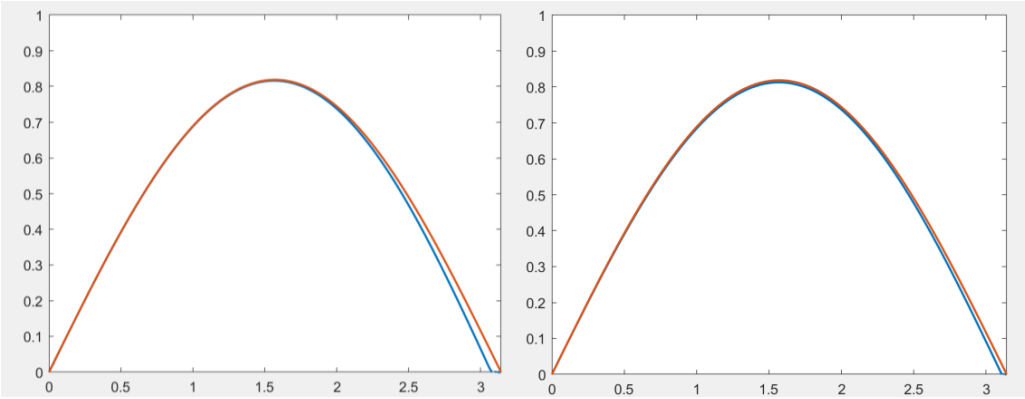
3、基本结果

考察一个简单的例子：

$$u_t = u_{xx}, u(0,t) = u(\pi,t) = 0, u(x,0) = \sin x$$

其精确解为$u(t,x) = e^{-t}\sin x$。

令时间步长为 1e-5，N = 3，m = 12，在层次 p = 5 与 6 迭代至 0.2 秒的结果如下：

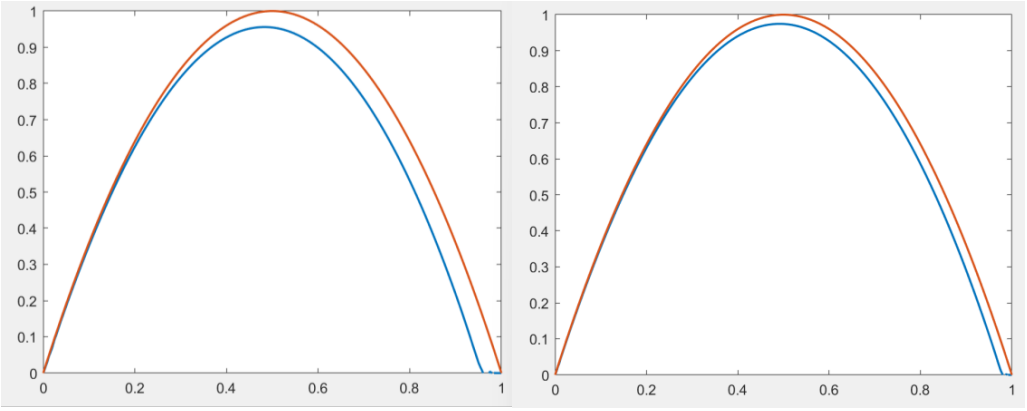


可以发现，在这种对 t 形式较好的情况下，可以得到十分接近精确（红线）的解。

而下面对精确解为解为$20t(1-x)x$的如下方程求解：

$$u_t - u_{xx} = 20(1-x)x + 40t, u(0,t) = u(1,t) = 0, u(x,0) = 0$$

由于此例子的数值性态不如前一个例子良好，时间步长需要更低以稳定更低，其他系数与之前相同，时间步长取$\frac{1}{30000}$，对 p = 4，5 可以得到：



4、迭代优化

为了更准确地进行常微分方程的估算，这里利用二阶龙格-库塔方法达到对时间步长的三阶误差。

设$c' = F(c,t)$，则二阶龙格-库塔的迭代过程为

$$F_1 = h_t F(c,t)$$

$$F_2 = h_t F(c + F_1/2, t + h_t/2)$$

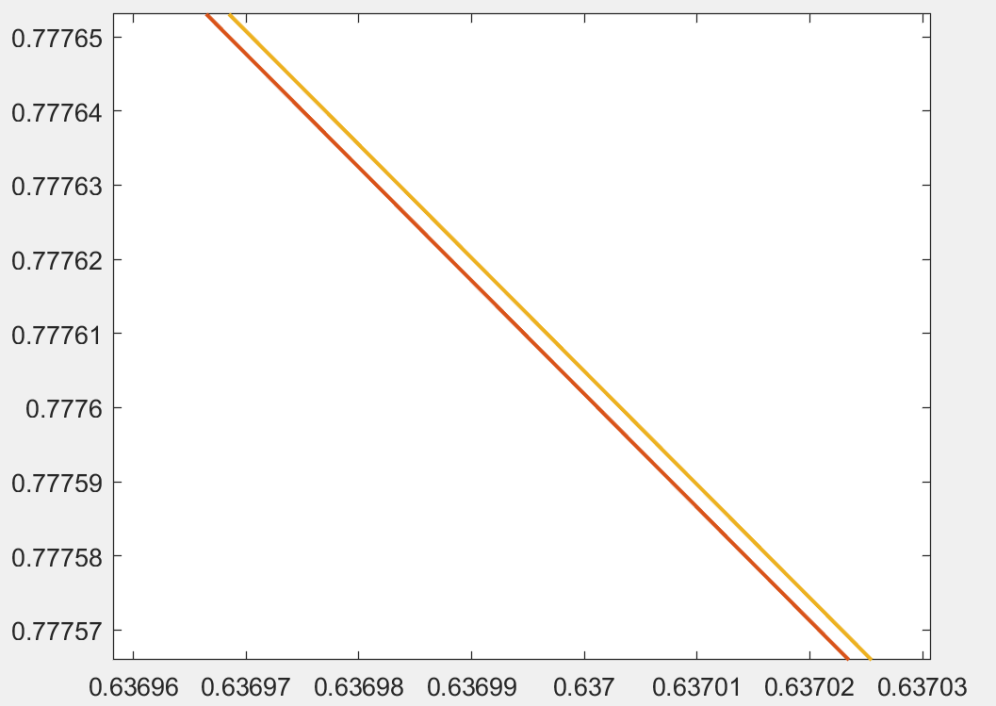
$$c = c + F_2$$

注意到-T * C + M \ B 的第一项只与 C 有关，第二项只与 t 有关，转化为代码即：

```
fnow = f(x, t) + df;
for i = 0:D-1
    count = fnow((i * 2^m + 1):((i + L) * 2^m + 1)) .* phi;
    B(i + 1) = fun.inte_4(count, i/2^p, (i+L)/2^p) * 2^(p/2);
end

F1 = ht * (-T * C + M \ B);

fnow = f(x, t + ht / 2) + df;
for i = 0:D-1
    count = fnow((i * 2^m + 1):((i + L) * 2^m + 1)) .* phi;
    B(i + 1) = fun.inte_4(count, i/2^p, (i+L)/2^p) * 2^(p/2);
end

F2 = ht * (-T * (C + F1 / 2) + M \ B);

C = C + F2;
```

由于对前两个性态相对较好的例子，采用优化迭代后的结果与原结果的差距在 ht 的量

级（对第二个例子取时间步长$\frac{1000}{3}, p = N = 3, t = 0.2$，放大后差距展示如下图），图上很难直观看出差距。由于随时间迭代运行时间较长，此处不进行更精细的比较。

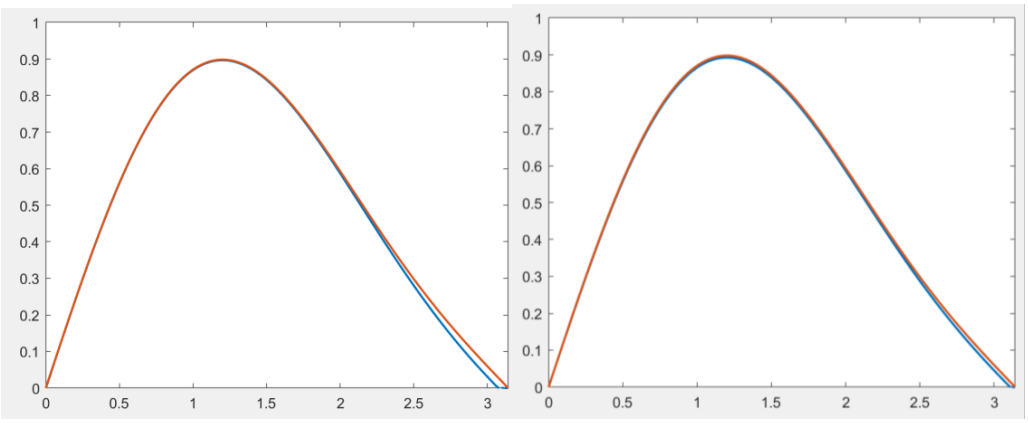


最后，考察一个无法直接分离变量的一般例子$\mathrm{u}(x,t) = e^{-t}\sin x + \mathrm{t}\sin 2\mathrm{x}$，其满足方程

$$u_t - u_{xx} = (4t+1)\sin 2x$$

$$\mathrm{u}(0,t) = \mathrm{u}(\pi,t) = 0, \mathrm{u}(x,0) = \sin x$$

在这个例子上，令时间步长为 `1e-5`，`N = 3`，`m = 12`，在层次 `p = 5` 与 6 迭代至 `0.2` 秒的结果如下：



结果里可以直观出展示此方法的逼近性能。并且作为显式方法，时间步长达到2^{-2p}就基本能保证数值稳定，稳定性也相对较好。通过此方法将 PDE 问题转化为 ODE 问题进行数值求解，可以比直接差分估算达到更好的效果。

三、总结

小波分析大作业 x

数值分析综合练习 ✓

好像数值分析也就学了插值、数值微分、数值积分、ODE 数值解，没想到能在这个大作业全用一遍。代码看起来不多，但绝大部分时间都花在了计算下标关系和尝试各种处理方式的时间，大概是 1 小时报告对应 10 行代码吧。

总之又一门课完结了，感谢老师助教这学期的付出！

附录-文件列表

est.m	数值微分/积分估算
res.gif	PDE 例 3 的 gif 变化展示
show_ode1.m	放缩[0, 1]区间求解例 1
show_ode2.m	放缩[0, 1]区间求解例 2
show_ode3.m	放缩[0, 1]区间求解例 3
show_ode_err1.m	放缩[0, 1]区间对例 1 误差估计
show_ode_err2.m	放缩[0, 1]区间在二阶公式下对例 1 误差估计
show_ode_expand1.m	放缩[0, 2N-1]区间求解例 1
show_ode_expand2.m	放缩[0, 2N-1]区间对例 1 误差估计
show_pde1.m	欧拉迭代求解 PDE 例 1
show_pde2.m	欧拉迭代求解 PDE 例 2
show_pde3.m	欧拉迭代与 RK 迭代对比
show_pde4.m	RK 迭代求解 PDE 例 3
wavelet_PDE.m	包含二阶 RK 迭代的 PDE 求解
wavelet_PDE_simple.m	包含欧拉迭代的 PDE 求解
wavelet_solve.m	放缩至[0, 1]区间求解 ODE
wavelet_solve2.m	放缩至[0, 1]区间并用二阶误差公式求解 ODE
wavelet_solve_expand.m	放缩至[0, 2N-1]区间求解 ODE