Numerical Simulation of Dynamic Systems XI

Prof. Dr. François E. Cellier
Department of Computer Science
ETH Zurich

April 9, 2013

The Method of Lines

Until now, we have dealt with *ordinary differential equations (ODEs)* only. We would now like to extend the discussion to *partial differential equations (PDEs)* as well.

Given the PDE that describes the diffusion of heat in a single space dimension:

$$\frac{\partial u}{\partial t} = \sigma \cdot \frac{\partial^2 u}{\partial x^2}$$

We may discretize the space variable:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_i} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\delta x^2}$$

while keeping the time variable continuous.

δx is the (here equidistantly chosen) distance between two neighboring discretization points in space, i.e., the so-called *grid width* of the discretization.

The Method of Lines II

In this way, we convert the former PDE to a set of ODEs that approximate the PDE:

$$\frac{du_i}{dt} \approx \sigma \cdot \frac{u_{i+1} - 2u_i + u_{i-1}}{\delta x^2}$$

The resulting ODEs can now be simulated using any of the numerical ODE solvers introduced in the earlier presentations.

Differentiation in Space

It makes sense to differentiate the solution function of the PDE with respect to the space variable using the same order of approximation accuracy that we use for the integration over time. To this end, we can once again make use of *Newton-Gregory polynomials*.

For example, if we like to obtain an approximation of fourth order of the second spatial derivative, we can formulate a polynomial of fourth order that passes through the five support values x_{i-2}, x_{i-1}, x_i, x_{i+1}, and x_{i+2}.

We use a backward Newton-Gregory polynomial around the point x_{i+2}:

$$u(x) = u_{i+2} + s\nabla u_{i+2} + \left(\frac{s^2}{2} + \frac{s}{2}\right)\nabla^2 u_{i+2} + \left(\frac{s^3}{6} + \frac{s^2}{2} + \frac{s}{3}\right)\nabla^3 u_{i+2} + \ldots$$

Therefore:

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{\delta x^2}\left[\nabla^2 u_{i+2} + (s+1)\nabla^3 u_{i+2} + \left(\frac{s^2}{2} + \frac{3s}{2} + \frac{11}{12}\right)\nabla^4 u_{i+2} + \ldots\right]$$

Differentiation in Space II

We evaluate at $x = x_i$, i.e., at $s = -2$ and truncate the ∇ operator after the fourth-order term:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_i} \approx \frac{1}{12\delta x^2}\left(-u_{i+2} + 16u_{i+1} - 30u_i + 16u_{i-1} - u_{i-2}\right)$$

Third-order methods must pass through four points. Hence the support values can no longer be placed symmetrically around the evaluation point. For example, we might try to obtain a method that passes through the four points x_{i-1}, x_i, x_{i+1}, and x_{i+2}.

Identifying the method parameters, we obtain:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_i} \approx \frac{1}{\delta x^2}\left(0u_{i+2} + u_{i+1} - 2u_i + u_{i-1}\right)$$

We conclude that the symmetric method of order 2 is in reality a method of order 3.

This is no accident. Symmetric methods always gain one additional order of approximation accuracy.

Differentiation in Space III

To obtain a fourth-order accurate method for $\partial^2 u/\partial x^2$ at x_3, we can either develop a backward Newton-Gregory polynomial around the point x_5 and evaluate it at $s = -2$, or alternatively, we can develop a Newton-Gregory forward polynomial around the point x_1 and evaluate it at $s = +2$. The resulting formulae will be identical, and they will, in fact, be fifth-order accurate due to symmetry.

Unfortunately, we cannot use symmetric methods in the vicinity of the domain boundaries. For example, to obtain a fourth-order accurate method for $\partial^2 u/\partial x^2$ at x_2, we can either develop a backward Newton-Gregory polynomial around the point x_5 and evaluate it at $s = -3$, or alternatively, we can develop a Newton-Gregory forward polynomial around the point x_1 and evaluate it at $s = +1$. The resulting formulae will again be identical, but they are no longer symmetric around the point x_2. Hence, we no longer gain the additional order of approximation accuracy. If we really want a fifth-order method, we should use an additional support value at x_6.

Differentiation in Space IV

Let us assume our domain contains 11 points in space and we wish to develop a fourth-order method. Then, the following formulae will need to be used in the vicinity of the two borders:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_1} = \frac{1}{12\delta x^2}\left(11u_5 - 56u_4 + 114u_3 - 104u_2 + 35u_1\right)$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_2} = \frac{1}{12\delta x^2}\left(-u_5 + 4u_4 + 6u_3 - 20u_2 + 11u_1\right)$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_{10}} = \frac{1}{12\delta x^2}\left(11u_{11} - 20u_{10} + 6u_9 + 4u_8 - u_7\right)$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_{11}} = \frac{1}{12\delta x^2}\left(35u_{11} - 104u_{10} + 114u_9 - 56u_8 + 11u_7\right)$$

Since most of the points are internal to the domain, odd-order approximations are usually preferred over even-order approximations.

Boundary Conditions

We need to talk about the topic of *boundary conditions*. Each PDE is accompanied not only by initial conditions, but also by boundary conditions.

The heat equation in one space dimension, $x \in [0.0, 1.0]$, requires two boundary conditions, e.g.:

$$u(x = 0.0, t) = 100.0$$
$$\frac{\partial u}{\partial x}(x = 1.0, t) = 0.0$$

The simplest type of boundary conditions is the *Dirichlet condition* that imposes at the border a value on the variable u. In this case, we need to eliminate the differential equation at that point and replace it by an algebraic equation. In the above example:

$$u_1 = 100.0$$

Boundary Conditions II

Another simple type of boundary conditions is the *symmetrical Neumann condition* that imposes at the border a value of zero on the variable $\frac{\partial u}{\partial x}$.

In this case, we can use a simple trick. We double the domain, i.e., in the above example, we would enlarge the domain in space to $x \in [0.0, 2.0]$ and apply the same boundary condition at the point $x = 2.0$ as at the point $x = 0.0$:

$$u_{21} = 100.0$$

For symmetry reasons, we obtain implicitly at the center the condition:

$$\frac{\partial u_{11}}{\partial x} = 0.0$$

Yet, it is not necessary to formulate this condition explicitly.

Boundary Conditions III

Therefore we can use symmetrical formulae at the center:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_{10}} = \frac{1}{12\delta x^2}\left(-u_{12} + 16u_{11} - 30u_{10} + 16u_9 - u_8\right)$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_{11}} = \frac{1}{12\delta x^2}\left(-u_{13} + 16u_{12} - 30u_{11} + 16u_{10} - u_9\right)$$

and as we know, due to symmetry, that $u_{12} = u_{10}$ and $u_{13} = u_9$, we can rewrite the above equations as:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_{10}} = \frac{1}{12\delta x^2}\left(16u_{11} - 31u_{10} + 16u_9 - u_8\right)$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_{11}} = \frac{1}{12\delta x^2}\left(-30u_{11} + 32u_{10} - 2u_9\right)$$

and now, we don't need the symmetrical points any longer, i.e., in reality, we don't need to double the domain.

Boundary Conditions IV

A third simple type of boundary conditions is the *temporal condition* that imposes at the border a value on the variable $\frac{\partial u}{\partial t}$:

$$\frac{\partial u}{\partial t}(x = 0.0, t) = f(t)$$

In this case, we replace the original differential equation at that point by a new one:

$$\dot{u}_1 = f(t)$$

Boundary Conditions V

A fourth type of boundary condition, that is a bit more general and difficult to treat, is the *Robin condition*:

$$g\left(u(x = 1.0, t)\right) + h\left(\frac{\partial u}{\partial x}(x = 1.0, t)\right) = f(t)$$

where f, g, and h are functions of time.

For example, we might have to deal with a boundary condition of the type:

$$\frac{\partial u}{\partial x}(x = 1.0, t) = -k \cdot \left(u(x = 1.0, t) - u_{\text{amb}}(t)\right)$$

where $u_{\text{amb}}(t)$ is the ambient temperature.

In this case, we replace the spatial derivative by an approximation using a Newton-Gregory polynomial:

$$\left.\frac{\partial u}{\partial x}\right|_{x=x_{11}} = \frac{1}{12\delta x}\left(25u_{11} - 48u_{10} + 36u_9 - 16u_8 + 3u_7\right)$$

Solving for the unknown u_{11}, we obtain an equivalent *Dirichlet condition*:

$$u_{11} = \frac{12k \cdot \delta x \cdot u_{\text{amb}} + 48u_{10} - 36u_9 + 16u_8 - 3u_7}{12k \cdot \delta x + 25}$$

Boundary Conditions VI

Sometimes, we need to deal with *non-linear boundary conditions*. For example:

$$\frac{\partial u}{\partial x}(x = 1.0, t) = -k \cdot \left(u(x = 1.0, t)^4 - u_{\text{amb}}(t)^4\right)$$

which can be converted to:

$$\begin{aligned}\mathcal{F}(u_{11}) =& 12k \cdot \delta x \cdot u_{11}^4 + 25u_{11} - 12k \cdot \delta x \cdot u_{\text{amb}}^4 - 48u_{10} + 36u_9 \\ & - 16u_8 + 3u_7 = 0.0\end{aligned}$$

In this way, we obtain an *implicitly formulated boundary condition* that can be solved by *Newton iteration*.

Boundary Conditions VII

Finally, we may want to consider situations where we have to deal with *separate neighboring space regions governed by different PDEs*.

For example, we may wish to simulate the diffusion of heat across two separate materials that are in contact with each other. We thus have two neighboring regions representing two different materials:

$$\begin{aligned}\frac{\partial u}{\partial t} &= \sigma_u \cdot \frac{\partial^2 u}{\partial x^2} \\ \frac{\partial v}{\partial t} &= \sigma_v \cdot \frac{\partial^2 v}{\partial x^2}\end{aligned}$$

where the PDE in u is valid in the region $x \in [0.0, 1.0]$, whereas the PDE in v is valid in the region $x \in [1.0, 1.1]$.

Boundary Conditions VIII

Now, we have to deal with *internal boundary conditions* at the common border between the two regions, e.g.:

$$\begin{aligned}\frac{\partial u}{\partial x}(x = 1.0, t) &= -k_u \cdot (u(x = 1.0, t) - v(x = 1.0, t)) \\ \frac{\partial v}{\partial x}(x = 1.0, t) &= -k_v \cdot (v(x = 1.0, t) - u(x = 1.0, t))\end{aligned}$$

which can be transformed to an equivalent set of linear algebraic equations:

$$\begin{aligned}(12k_u \cdot \delta x_u + 25)u_{11} - 12k_u \cdot \delta x_u \cdot v_1 &= 48u_{10} - 36u_9 + 16u_8 - 3u_7 \\ -12k_v \cdot \delta x_v \cdot u_{11} + (12k_v \cdot \delta x_v + 3)v_1 &= 16v_2 - 36v_3 + 48u_4 - 25v_5\end{aligned}$$

These equations form together an *algebraic loop* that can be solved either symbolically or numerically.

Classification of PDEs

Some simple types of PDEs are so common that they were given special names. Let us consider the following PDE in two variables x and y:

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} = d$$

which is characteristic of many field problems in physics. x can be either spatial or temporal variables, and a, b, c, and d can be arbitrary functions of x, y, u, $\partial u/\partial x$, and $\partial u/\partial y$. Such a PDE is called *quasi-linear*, since it is linear in the highest derivatives.

Depending on the numerical relationship between a, b, and c, the above equation is classified as either being *parabolic*, *hyperbolic*, or *elliptic*. The classification is as follows:

$$\begin{aligned}b^2 - 4ac > 0 &\Longrightarrow \text{PDE is hyperbolic} \\ b^2 - 4ac = 0 &\Longrightarrow \text{PDE is parabolic} \\ b^2 - 4ac < 0 &\Longrightarrow \text{PDE is elliptic}\end{aligned}$$

This classification makes sense, since the numerical methods most suitable for these three types of PDEs are vastly different.

Parabolic PDEs

One of the simplest parabolic PDEs is the heat equation in one space dimension. Let us consider a complete example:

$$\frac{\partial u}{\partial t} = \frac{1}{10\pi^2} \cdot \frac{\partial^2 u}{\partial x^2} \quad ; \quad x \in [0,1] \quad ; \quad t \in [0,\infty)$$
$$u(x, t=0) = \cos(\pi \cdot x)$$
$$u(x=0, t) = \exp(-t/10)$$
$$\frac{\partial u}{\partial x}(x=1, t) = 0$$

This model can be converted to the following set of ODEs using the *method of lines*:

$$u_1 = \exp(-t/10)$$
$$\dot{u}_2 = \frac{n^2}{10\pi^2} \cdot (u_3 - 2u_2 + u_1)$$
$$\dot{u}_3 = \frac{n^2}{10\pi^2} \cdot (u_4 - 2u_3 + u_2)$$
$$\dots$$
$$\dot{u}_n = \frac{n^2}{10\pi^2} \cdot (u_{n+1} - 2u_n + u_{n-1})$$
$$\dot{u}_{n+1} = \frac{n^2}{5\pi^2} \cdot (-u_{n+1} + u_n)$$

initial conditions :

$$u_2(0) = \cos\left(\frac{\pi}{n}\right)$$
$$u_3(0) = \cos\left(\frac{2\pi}{n}\right)$$
$$\dots$$
$$u_n(0) = \cos\left(\frac{(n-1)\pi}{n}\right)$$
$$u_{n+1}(0) = \cos(\pi)$$

Parabolic PDEs II

This is a *linear, time-invariant, inhomogeneous, n^{th}-order, single-input system* of the type:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot u$$

where:

$$\mathbf{A} = \frac{n^2}{10\pi^2} \cdot \begin{pmatrix} -2 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 2 & -2 \end{pmatrix}$$

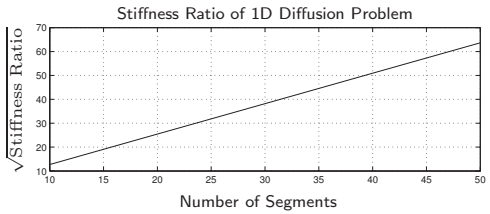
A is a *band-structured matrix* of dimensions $n \times n$. The *band width* of the matrix is three.

Parabolic PDEs III

We can look at the distribution of eigenvalues of the **A**-matrix in function of the spatial discretization:

$n = 3$	$n = 4$	$n = 5$	$n = 6$	$n = 7$
-0.0244	-0.0247	-0.0248	-0.0249	-0.0249
-0.1824	-0.2002	-0.2088	-0.2137	-0.2166
-0.3403	-0.4483	-0.5066	-0.5407	-0.5621
	-0.6238	-0.9884	-0.9183	-0.9929
		-0.8044	-1.2454	-1.4238
			-1.4342	-1.7693
				-1.9610

Therefore:



Parabolic PDEs IV

- We notice that all eigenvalues of the heat equation converted to a set of ODEs are on the negative real axis. The heat equation thus never leads to oscillations.
- We notice further that, whereas the slowest eigenvalue of the converted ODE set remains more or less at the same location, ever faster (more heavily damped) eigenvalues are added as the number of discretization points grows, i.e., as the discretization grid is made finer.
- **Parabolic PDEs converted to a set of equivalent ODEs using the method of lines always lead to stiff systems.**
- **Hence there are now two types of stiff systems: lumped parameter systems that are naturally stiff, and parabolic PDEs that become stiff in the conversion to ODEs.**

The Consistency Error

It may be interesting to analyze, how well the discretized system, described by a set of ODEs approximates the original continuous system that is described by a PDE.

To this end, we shall compare the *analytical solutions* of the two problems to each other.

Luckily, we chose a PDE problem, the analytical solution of which is known:

$$u_c(x,t) = \exp(-t/10) \cdot \cos(\pi \cdot x)$$

The discretized problem is a bit harder to handle. We converted the PDE to a continuous linear time-invariant system of the form:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot u$$
$$\mathbf{y} = \mathbf{C} \cdot \mathbf{x} + \mathbf{d} \cdot u$$

In **Matlab**, we can make a continuous-time system description out of the four matrices:

```
Sc = ss(A, b, C, d);
```


The Consistency Error II

The continuous-time system can be converted to an equivalent discrete-time system of the form:

$$\mathbf{x}_{k+1} = \mathbf{F} \cdot \mathbf{x}_k + \mathbf{g} \cdot u_k \quad (1)$$
$$\mathbf{y}_k = \mathbf{H} \cdot \mathbf{x}_k + \mathbf{i} \cdot u_k \quad (2)$$

using the Matlab statement:

```
Sd = c2d(Sc, h);
```

from which the **F**-matrix and **g**-vector of the discrete state equations can be extracted using the Matlab statement:

```
[F, g] = ssdata(Sd);
```

We define the *consistency error* as the difference between the analytical solutions of the distributed parameter system, u_c, and the discretized ODE model, u_d:

$$err = u_c - u_d$$

The Consistency Error III

The consistency error is defined for each grid point and for each time step, i.e., it is a function of time and space:

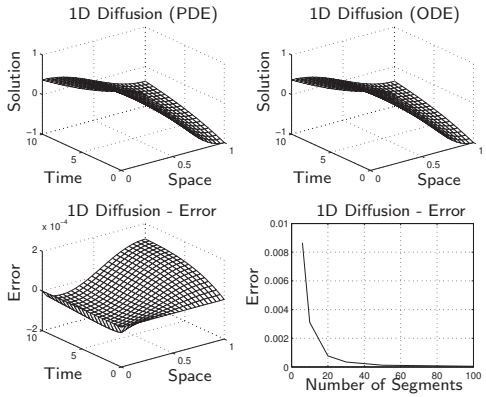
$$err = \mathrm{err}(x,t)$$

Yet, it makes also sense to look at the infinity norm of the consistency error. In Matlab:

```
ermax = max(max(abs(err)));
```


The Consistency Error IV

We display on one graph the analytical solution of the PDE problem, u_c, the analytical solution of the ODE problem, u_d, the consistency error, *err*, as well as the maximal consistency error, *ermax*, plotted against the number of discretization points in space.



The Consistency Error V

Is it possible to overcome the consistency error by using more accurate discretization formulae in space?

$$
\begin{aligned}
u_1 &= \exp(-t/10) \\
\dot{u}_2 &= \frac{n^2}{120\pi^2} \cdot (u_6 - 6u_5 + 14u_4 - 4u_3 - 15u_2 + 10u_1) \\
\dot{u}_3 &= \frac{n^2}{120\pi^2} \cdot (-u_5 + 16u_4 - 30u_3 + 16u_2 - u_1) \\
\dot{u}_4 &= \frac{n^2}{120\pi^2} \cdot (-u_6 + 16u_5 - 30u_4 + 16u_3 - u_2) \\
&\dots \\
\dot{u}_{n-1} &= \frac{n^2}{120\pi^2} \cdot (-u_{n+1} + 16u_n - 30u_{n-1} + 16u_{n-2} - u_{n-3}) \\
\dot{u}_n &= \frac{n^2}{120\pi^2} \cdot (16u_{n+1} - 31u_n + 16u_{n-1} - u_{n-2}) \\
\dot{u}_{n+1} &= \frac{n^2}{60\pi^2} \cdot (-15u_{n+1} + 16u_n - u_{n-1})
\end{aligned}
$$

The Consistency Error VI

The **A**-matrix now takes the form:

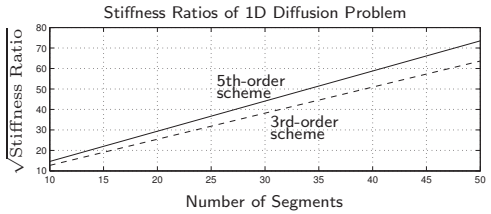
$$
\mathbf{A} = \frac{n^2}{120\pi^2} \cdot \begin{pmatrix}
-15 & -4 & 14 & -6 & 1 & \dots & 0 & 0 & 0 & 0 \\
16 & -30 & 16 & -1 & 0 & \dots & 0 & 0 & 0 & 0 \\
-1 & 16 & -30 & 16 & -1 & \dots & 0 & 0 & 0 & 0 \\
0 & -1 & 16 & -30 & 16 & \dots & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & \dots & 16 & -30 & 16 & -1 \\
0 & 0 & 0 & 0 & 0 & \dots & -1 & 16 & -31 & 16 \\
0 & 0 & 0 & 0 & 0 & \dots & 0 & -2 & 32 & -30
\end{pmatrix}
$$

with the eigenvalue distribution:

$n = 5$	$n = 6$	$n = 7$	$n = 8$	$n = 9$
-0.0250	-0.0250	-0.0250	-0.0250	-0.0250
-0.2288	-0.2262	-0.2253	-0.2251	-0.2250
-0.5910	-0.6414	-0.6355	-0.6302	-0.6273
-0.7654	-0.9332	-1.1584	-1.2335	-1.2368
-1.2606	-1.3529	-1.4116	-1.6471	-1.9150
	-1.8671	-2.0761	-2.1507	-2.2614
		-2.5770	-2.9084	-3.0571
			-3.3925	-3.8460
				-4.3147

The Consistency Error VII

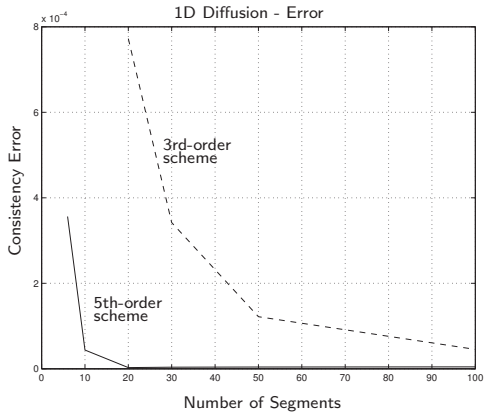
Comparing the stiffness ratios of the two methods of orders 3 and 5:



The stiffness ratio isn't much influenced by the order of approximation accuracy of the spatial discretization scheme.

The Consistency Error VIII

Comparing the maximal consistency errors of the two methods of orders 3 and 5:



The consistency error is very much influenced by the order of approximation accuracy of the spatial discretization scheme.

The Consistency Error IX

- The biggest crux in the numerical solution of parabolic PDE problems is the following. If we double the number of segments, the number of ODEs to be simulated doubles as well. However, since the stiffness ratio grows quadratically in the number of segments, the step size needs to decrease inverse quadratically in order to preserve the same accuracy. Hence doubling the number of segments forces us to quadruple the number of time steps.
- **The simulation effort grows cubically in the number of segments.**
- Luckily, the higher-order discretization formulae enable us to get away with a smaller number of discretization points, i.e., use a coarser grid.
- **Using a 5^{th}-order discretization scheme usually pays off.**

Richardson Extrapolation

The order of approximation accuracy of the spatial differentiation would not be all that important if only we could use a very narrow grid.

Hence we try once again the idea of the *Richardson extrapolation*, but this time across space instead of time.

We shall work with 4 different approximations of order 3:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_i}^{P_1}(\delta x^2) = \frac{u_{i+1} - u_i + u_{i-1}}{\delta x^2}$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_i}^{P_2}(4\delta x^2) = \frac{u_{i+2} - u_i + u_{i-2}}{4\delta x^2}$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_i}^{P_3}(9\delta x^2) = \frac{u_{i+3} - u_i + u_{i-3}}{9\delta x^2}$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_i}^{P_4}(16\delta x^2) = \frac{u_{i+4} - u_i + u_{i-4}}{16\delta x^2}$$

These approximations differ only in the grid width δx used to obtain them.

Richardson Extrapolation II

We develop these approximations into Taylor series around the correct, but unknown, second derivative, $\frac{\partial^2 u}{\partial x^2}$:

$$\frac{\partial^2 u}{\partial x^2}(\eta) = \frac{\partial^2 u}{\partial x^2} + e_1 \cdot \eta + e_2 \cdot \frac{\eta^2}{2!} + e_3 \cdot \frac{\eta^3}{3!} + \dots$$

Therefore:

$$\frac{\partial^2 u}{\partial x^2}^{P_1}(\delta x^2) \approx \frac{\partial^2 u}{\partial x^2} + e_1 \cdot \delta x^2 + \frac{e_2}{2!} \cdot \delta x^4 + \frac{e_3}{3!} \cdot \delta x^6$$
$$\frac{\partial^2 u}{\partial x^2}^{P_2}(4\delta x^2) \approx \frac{\partial^2 u}{\partial x^2} + e_1 \cdot (4\delta x^2) + \frac{e_2}{2!} \cdot (4\delta x^2)^2 + \frac{e_3}{3!} \cdot (4\delta x^2)^3$$
$$\frac{\partial^2 u}{\partial x^2}^{P_3}(9\delta x^2) \approx \frac{\partial^2 u}{\partial x^2} + e_1 \cdot (9\delta x^2) + \frac{e_2}{2!} \cdot (9\delta x^2)^2 + \frac{e_3}{3!} \cdot (9\delta x^2)^3$$
$$\frac{\partial^2 u}{\partial x^2}^{P_4}(16\delta x^2) \approx \frac{\partial^2 u}{\partial x^2} + e_1 \cdot (16\delta x^2) + \frac{e_2}{2!} \cdot (16\delta x^2)^2 + \frac{e_3}{3!} \cdot (16\delta x^2)^3$$

Richardson Extrapolation III

In matrix-vector notation:

$$\begin{pmatrix} \frac{\partial^2 u}{\partial x^2}^{P_1} \\ \frac{\partial^2 u}{\partial x^2}^{P_2} \\ \frac{\partial^2 u}{\partial x^2}^{P_3} \\ \frac{\partial^2 u}{\partial x^2}^{P_4} \end{pmatrix} \approx \begin{pmatrix} (\delta x^2)^0 & (\delta x^2)^1 & (\delta x^2)^2 & (\delta x^2)^3 \\ (4\delta x^2)^0 & (4\delta x^2)^1 & (4\delta x^2)^2 & (4\delta x^2)^3 \\ (9\delta x^2)^0 & (9\delta x^2)^1 & (9\delta x^2)^2 & (9\delta x^2)^3 \\ (16\delta x^2)^0 & (16\delta x^2)^1 & (16\delta x^2)^2 & (16\delta x^2)^3 \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial^2 u}{\partial x^2} \\ e_1 \\ e_2/2 \\ e_3/6 \end{pmatrix}$$

Thus:

$$\frac{\partial^2 u}{\partial x^2} \approx \begin{pmatrix} \frac{56}{35} & -\frac{28}{35} & \frac{8}{35} & -\frac{1}{35} \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial^2 u}{\partial x^2}^{P_1} \\ \frac{\partial^2 u}{\partial x^2}^{P_2} \\ \frac{\partial^2 u}{\partial x^2}^{P_3} \\ \frac{\partial^2 u}{\partial x^2}^{P_4} \end{pmatrix}$$

Once again, the top row of the inverse of the Vandermonde matrix does not depend on δx.

Richardson Extrapolation IV

By substituting the four approximations into the formula for $\frac{\partial^2 u}{\partial x^2}$, we obtain:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_i} \approx \frac{1}{5040\delta x^2}(-9u_{i+4} + 128u_{i+3} - 1008u_{i+2} + 8064u_{i+1} - 14350u_i + 8064u_{i-1} - 1008u_{i-2} + 128u_{i-3} - 9u_{i-4})$$

which is exactly the *central 9^{th}-order accurate approximation of the second spatial derivative*.

Richardson Extrapolation V

- **Once again, Richardson extrapolation has maximized the order of approximation accuracy of the method.**
- The *Richardson coefficients* are different here from those found in the case of time integration using Richardson extrapolation. This is due to the quadratic δx^2 used in our new application in comparison with the linear h used in the time discretization.
- Each additional approximation used increases the order of the overall method by two, rather than by one, as each additional approximation adds two support values to the set, one on each side.
- Since we didn't state in the derivation of the Richardson extrapolation, how the individual spatial derivatives are being computed, the approach works for all approximations. In particular, it also works for the biased formulae used in the vicinity of the boundaries. The Richardson coefficients do not change.

Order and Grid Width Control

Let us now look at a slightly different problem:

$$\frac{\partial u}{\partial t} = 4\frac{\partial^2 u}{\partial x^2} \quad ; \quad x \in [0,1] \quad ; \quad t \in [0,\infty)$$
$$u(x, t=0) = 20\sin\left(\frac{\pi}{2}x\right) + 300$$
$$u(x=0, t) = 20\sin\left(\frac{\pi}{12}t\right) + 300$$
$$\frac{\partial u}{\partial x}(x=1, t) = 0$$

- We again solve a one-dimensional heat equation, but with a different time constant, and different initial and boundary conditions.
- This time around, we don't know the analytical solution, hence we cannot compute the consistency error explicitly.
- Similarly to the step-size control algorithms discussed in the previous chapters, we need an *estimator of the spatial discretization error*.
- All numerical algorithms should have a second algorithm built in to them that reasons about the sanity of the first algorithm and starts screaming if it thinks that something is going awry. Without such a sanity check, numerical algorithms are never safe.

Order and Grid Width Control II

We propose to compute all spatial derivatives twice, once with the grid size δx, and once with the grid size $2\delta x$ using central differences:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|^{P_1}_{x=x_i}(\delta x^2) = \frac{u_{i+1} - u_i + u_{i-1}}{\delta x^2}$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|^{P_2}_{x=x_i}(4\delta x^2) = \frac{u_{i+2} - u_i + u_{i-2}}{4\delta x^2}$$

The two approximations form two separate partial derivative vectors, $\mathbf{u}_{xx}^{P_1}$ and $\mathbf{u}_{xx}^{P_2}$. Using these approximations, we can formulate a spatial error estimate:

$$\varepsilon_{\text{rel}} = \frac{|\mathbf{u}_{xx}^{P_1} - \mathbf{u}_{xx}^{P_2}|}{\max(|\mathbf{u}_{xx}^{P_1}|, |\mathbf{u}_{xx}^{P_2}|, \delta)}$$

where δ is a fudge factor, e.g., $\delta = 10^{-10}$.

If the estimated spatial discretization error is too big, we must either choose a more narrow grid, or alternatively, we must increase the approximation order of the spatial derivatives.

Order and Grid Width Control III

Is it wasteful to compute the entire vector of spatial derivatives twice?

This question must clearly be answered in the negative. The two predictors can be used in a *Richardson corrector* step:

$$\mathbf{u}_{xx}^{C} = \frac{4}{3} \cdot \mathbf{u}_{xx}^{P_1} - \frac{1}{3} \cdot \mathbf{u}_{xx}^{P_2}$$

This is equivalent to having raised the approximation order of the spatial derivatives from three to five.

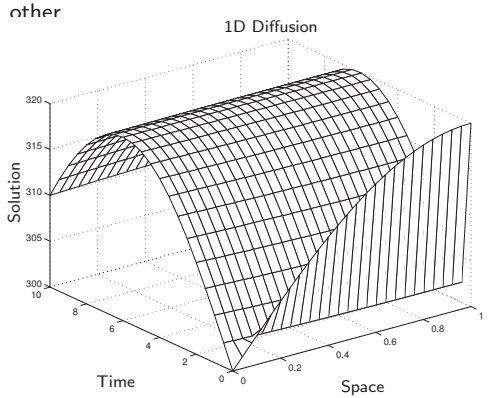
However, by writing the 5^{th}-order accurate spatial derivative formula in this way, we get an error estimator essentially for free.

The Startup Problem

Since parabolic PDEs converted to sets of ODEs by the method of lines invariably lead to *stiff systems*, we need to use a stiff system solver for their simulation. Traditionally, we will choose a **BDF algorithm**.

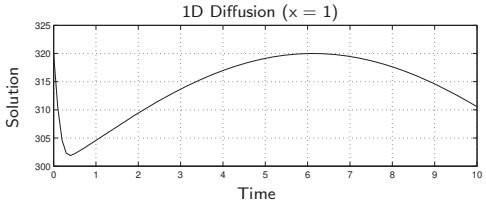
The problem is that BDF algorithms (except for BDF1 = BE) are not self-starting. Hence we need a starter algorithm. We now wish to compare different starter algorithms to each other.

Let us simulate:



The Startup Problem II

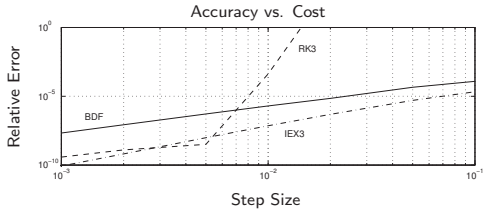
Let us look at a slice through the solution. We plot $u(x = 1, t)$:



The solution has a fast gradient at the beginning. Hence the choice of a suitable starter algorithm may be important.

The Startup Problem III

We compare three different starter algorithms and plot the accuracy vs. the cost (step size).



All three solutions were simulated using **BDF3** using a fixed step size. They only differ in the first two steps that use different startup algorithms employing the same step size used afterwards by BDF3:

1. BDF: We used one step of BDF1 (BE) followed by one step of BDF2.
2. RK3: We used two steps of FRK3.
3. IEX3: We used two steps of IEX3.

The Startup Problem IV

- The *BDF starter* didn't work very well. The simulation accumulates lots of errors during the first two step due to the low resolution of the low-order algorithms, and it doesn't recover from those errors ever.
- The *RK3 starter* worked much better, at least for sufficiently small step sizes. For larger step sizes, the RK3 algorithm loses its numerical stability, and consequently, accumulates lots of errors during the first two steps.
- The *IEX3 starter* worked best. It is a stiff system solver, and although the first two steps are relatively expensive to compute, this doesn't matter in the longer run.

I also tried a *BI*$4/5_{0.45}$ *starter*. It didn't work well on this example. The reason is that the backward RKF semi-step is numerically highly unstable. It is only stabilized by the Newton iteration. In the given application, we ran into roundoff error problems. The unstable semi-step produced numbers so big that the Newton iteration could not stabilize them any longer due to roundoff.

Conclusions

- In this presentation, we introduced the *method of lines* as a general means for converting PDE problems into equivalent ODE problems that we already know how to simulate.
- We looked at one class of PDE problems in detail, namely the class of *parabolic PDEs*, that always lead to *stiff systems* in their conversion to ODE systems.
- We also looked at a new type of error, the *consistency error*, that describes the difference between the analytical solutions of the original PDE problem and the converted ODE problem.
- We finally discussed an interesting algorithm for *grid width and order control* in spatial discretization and looked once more at the *startup problem* of stiff system solvers.