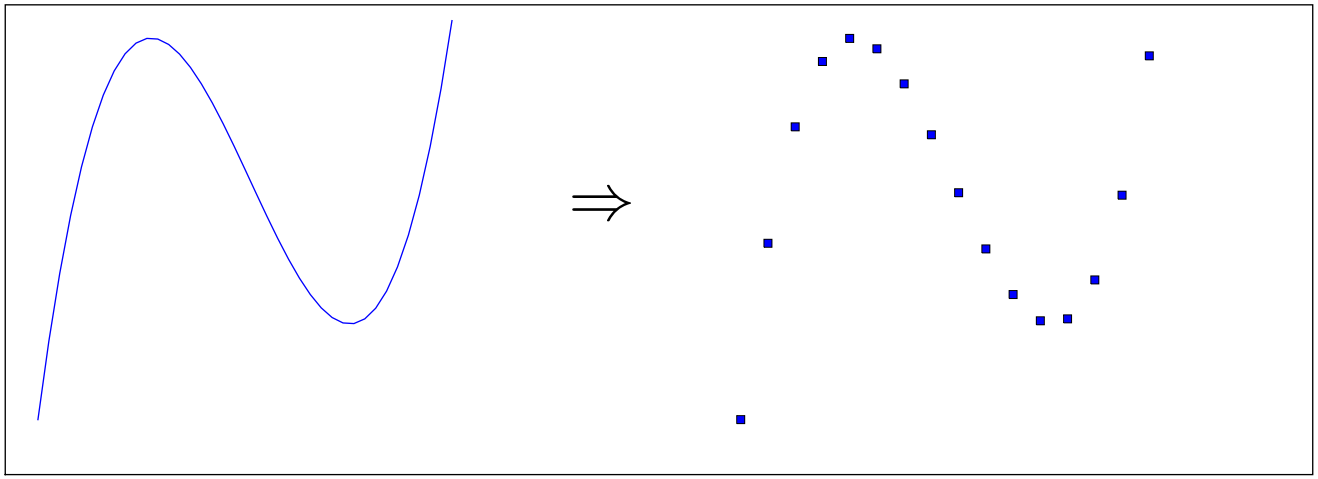
From Models to Algorithms



$$\frac{dy}{dt} \quad \rightarrow \quad \frac{y_{n+1} - y_n}{\Delta t}$$

$$f(x) = 0 \quad \rightarrow \quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Functions on the Computer

$$f(x) = \sin(x) \quad x \in [0, \pi]$$

Problem:

Infinite number of values of x.
How to store $f(x)$?
Computers are finite.

Solutions:

1. Compute $\sin(x)$ when you need the value for a specific x.
2. Store $f(x)$ in a number of points and then interpolate in other points.

The Sine Function

Suppose we want to generate a plot of the sine function for values of x between 0 and π. To this end, we define a set of x-values and an associated set of values of the sine function. More precisely, we define $n + 1$ points by

$$x_i = ih \text{ for } i = 0, 1, \ldots, n$$

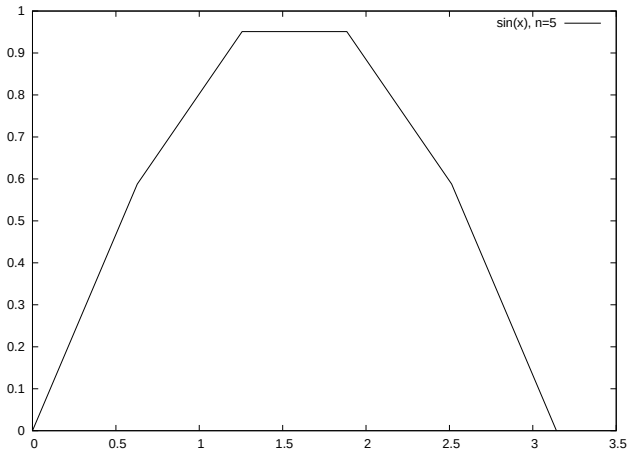
where $h = \pi/n$ and $n \geqslant 1$ is an integer. The associated function values are defined as

$$s_i = \sin(x_i) \text{ for } i = 0, 1, \ldots, n.$$

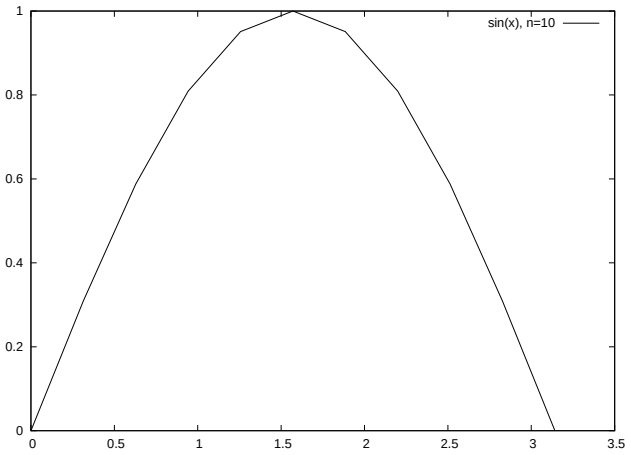
The Sine Function

```
from scitools.std import *

n = int(sys.argv[1])

x = linspace(0, pi, n+1)
s = sin(x)
plot(x, s, legend='sin(x), n=%d' % n, hardcopy='tmp.eps')
```

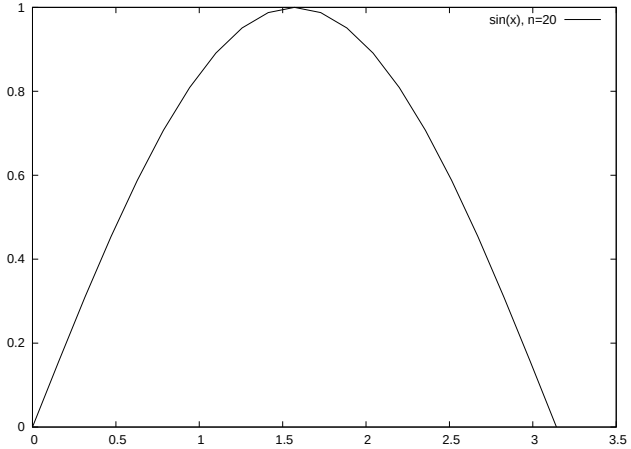
The Sine Function



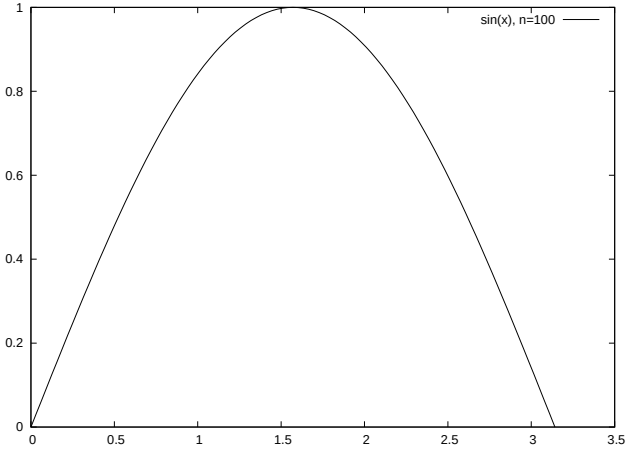
$n = 5$



$n = 10$



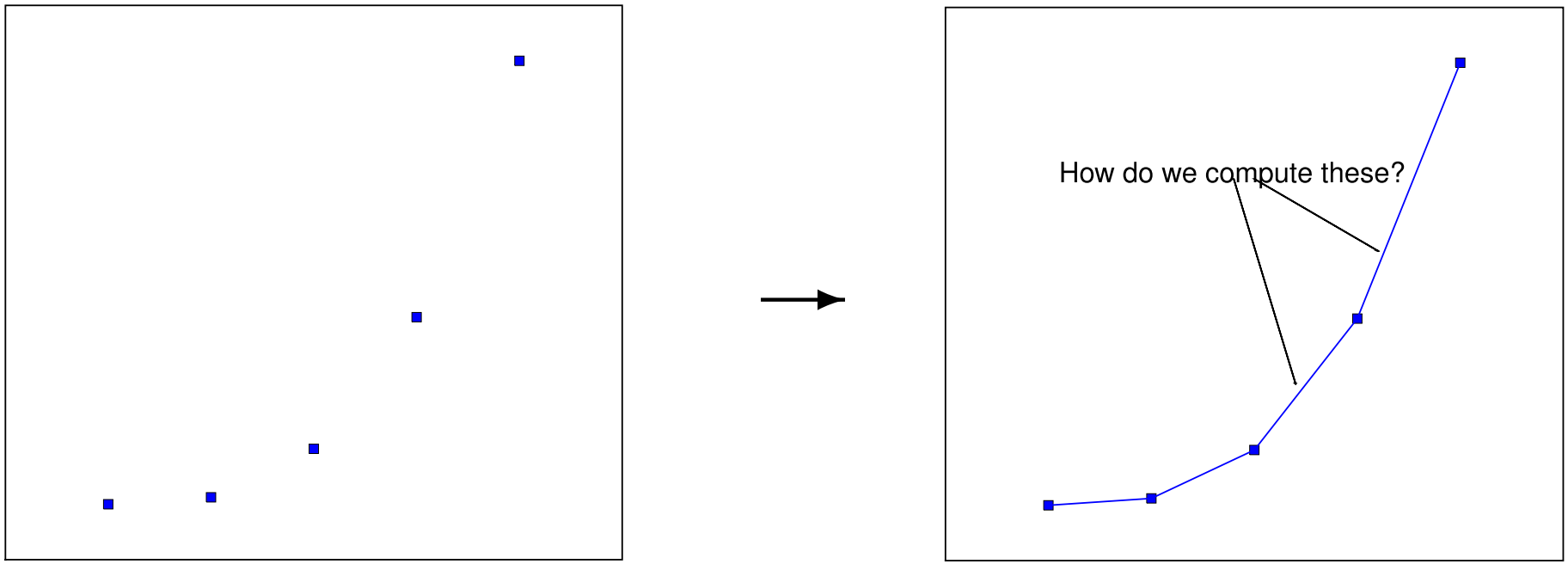
$n = 20$



$n = 100$

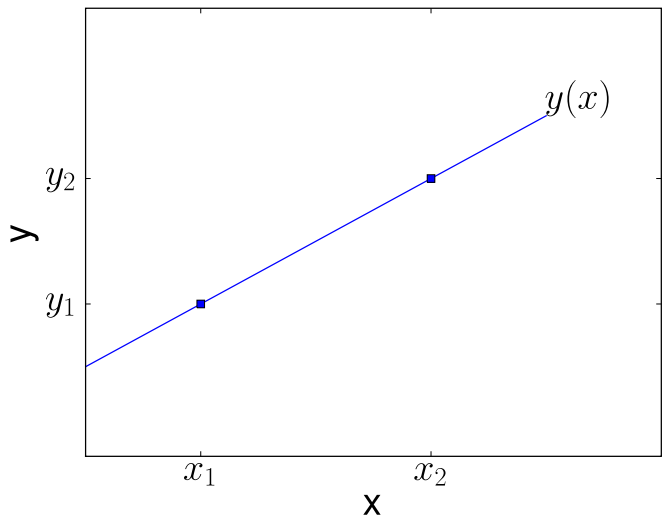
Plots of $\sin(x)$ with various n.

Interpolation



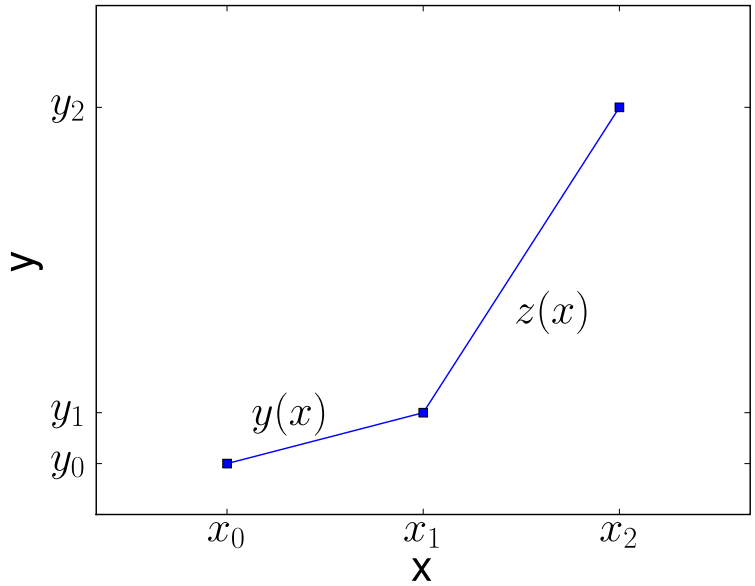
Interpolation

How do we compute values of a straight line between two given points?
Given (x_1, y_1) and (x_2, y_2):



$$y(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

How do we generalize this?



$$y(x) = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

$$z(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

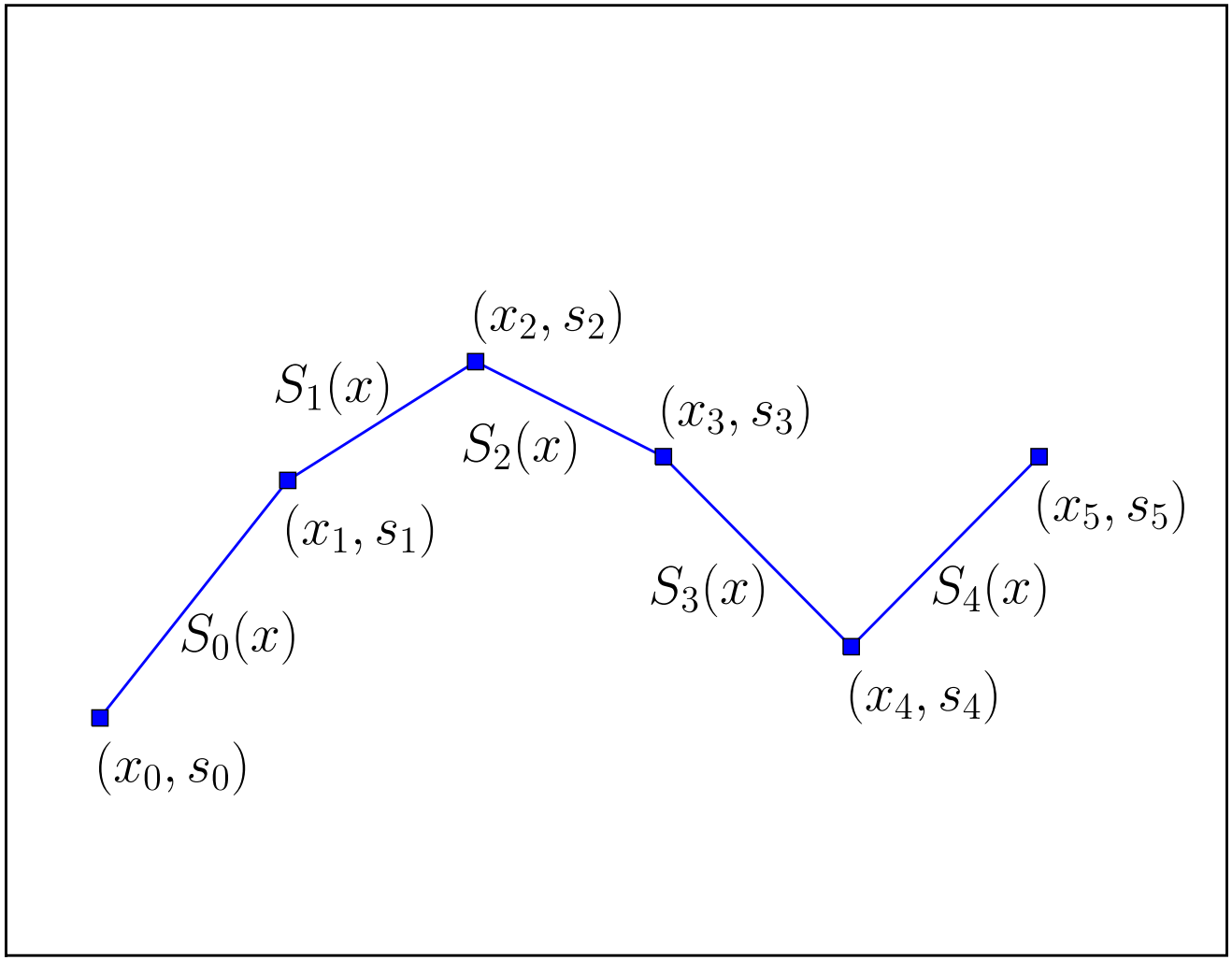
Interpolation

Assume that we know that a given x^* lies in the interval from $x = x_k$ to x_{k+1}, where the integer k is given. In the interval $x_k \leqslant x < x_{k+1}$, we define the linear function that passes through (x_k, s_k) and (x_{k+1}, s_{k+1}):

$$S_k(x) = s_k + \frac{s_{k+1} - s_k}{x_{k+1} - x_k}(x - x_k).$$

That is, $S_k(x)$ coincides with $\sin(x)$ at x_k and x_{k+1}, and between these nodes, $S_k(x)$ is linear. We say that $S_k(x)$ interpolates the discrete function $(x_i, s_i)_{i=0}^n$ on the interval $[x_k, x_{k+1}]$.

Interpolation



Interpolation

```
from numpy import *
import sys

xp = eval(sys.argv[1])
n = int(sys.argv[2])

def S_k(k):
    return s[k] + \
           ((s[k+1] - s[k])/(x[k+1] - x[k]))*(xp - x[k])
h = pi/n
x = linspace(0, pi, n+1)
s = sin(x)
k = int(xp/h)

print 'Approximation of sin(%s):  ' % xp, S_k(k)
print 'Exact value of sin(%s):    ' % xp, sin(xp)
print 'Eror in approximation:     ', sin(xp) - S_k(k)
```

Interpolation

To study the approximation, we put $x = \sqrt{2}$ and use the program `eval_sine.py` for $n = 5, 10$ and 20.

Terminal

```
eval_sine.py 'sqrt(2)' 5
Approximation of sin(1.41421356237):   0.951056516295
Exact value of sin(1.41421356237):     0.987765945993
Eror in approximation:       0.0367094296976
```

Terminal

```
eval_sine.py 'sqrt(2)' 10
Approximation of sin(1.41421356237):   0.975605666221
Exact value of sin(1.41421356237):     0.987765945993
Eror in approximation:       0.0121602797718
```

Interpolation

Terminal

```
eval_sine.py 'sqrt(2)' 20
Approximation of sin(1.41421356237):   0.987727284363
Exact value of sin(1.41421356237):     0.987765945993
Eror in approximation:       3.86616296923e-05
```

Note that the error is reduced as the n increases.

Differentiation Becomes Finite Differences

You have heard about derivatives. Probably, the following formulas are well known to you:

$$\frac{d}{dx}\sin(x) = \cos(x)$$
$$\frac{d}{dx}\ln(x) = \frac{1}{x}$$
$$\frac{d}{dx}x^m = mx^{m-1}$$

Differentiation Becomes Finite Differences

Why is differentiation so important? The reason is quite simple: The derivative is a mathematical expression of change. And change is, of course, essential in modeling various phenomena. If we know the state of a system, and we know the laws of change, then we can, in principle, compute the future of that system.

Differentiation Becomes Finite Differences

The mathematical definition of differentiation reads

$$f'(x) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon}\,.$$

Approximation:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

for small $h > 0$.

Differentiating the Sine Function

Consider $f(x) = \sin(x)$ and the associated derivative $f'(x) = \cos(x)$. If we put $x = 1$,we have

$$f'(1) = \cos(1) \approx 0.540\,,$$

and by putting $h = 1/100$ in (16) we get

$$f'(1) \approx \frac{f(1 + 1/100) - f(1)}{1/100} = \frac{\sin(1.01) - \sin(1)}{0.01} \approx 0.536\,.$$

Differentiating the Sine Function

```
def diff(f, x, h):
    return (f(x+h) - f(x))/float(h)

from math import *
import sys

x = eval(sys.argv[1])
h = eval(sys.argv[2])

approx_deriv = diff(sin, x, h)
exact = cos(x)
print 'The approximated value is: ', approx_deriv
print 'The correct value is:      ', exact
print 'The error is:              ', exact - approx_deriv
```

Running the program for $x = 1$ and $h = 1/1000$ gives

Terminal

```
forward_diff.py 1 0.001
The approximated value is:  0.53988148036
The correct value is:       0.540302305868
The error is:               0.00042082550781 3
```

Differences on a Mesh

Frequently, we will need finite difference approximations to a discrete function defined on a mesh. Suppose we have a discrete representation of the sine function: $(x_i, s_i)_{i=0}^{n}$. We want to compute approximations to the derivative of the sine function at the nodes in the mesh. Since we only have function values at the nodes, the h must be the difference between nodes, i.e., $h = x_{i+1} - x_i$. At node x_i we then have the following approximation of the derivative:

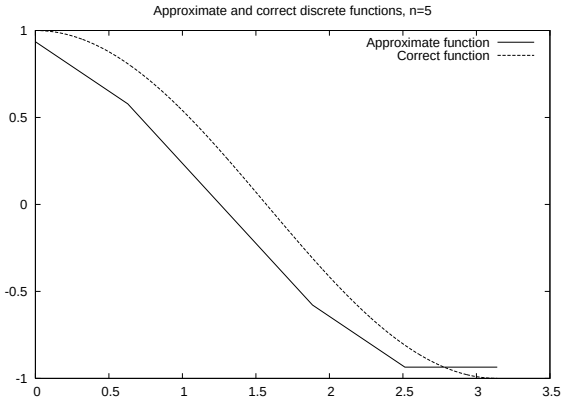
$$z_i = \frac{s_{i+1} - s_i}{h},$$

for $i = 0, 1, \ldots, n-1$.

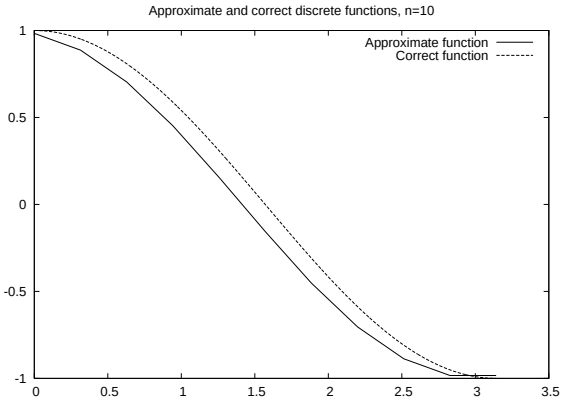
Differences on a Mesh

```
from scitools.std import *

n = int(sys.argv[1])

h = pi/n
x = linspace(0, pi, n+1)
s = sin(x)
z = zeros(len(s))
for i in xrange(len(z)-1):
    z[i] = (s[i+1] - s[i])/h
# special formula for end point_
z[-1] = (s[-1] - s[-2])/h
plot(x, z)

xfine = linspace(0, pi, 1001) # for more accurate plot
exact = cos(xfine)
hold()
plot(xfine, exact)
legend('Approximate function', 'Correct function')
title('Approximate and discrete functions, n=%d' % n)
```

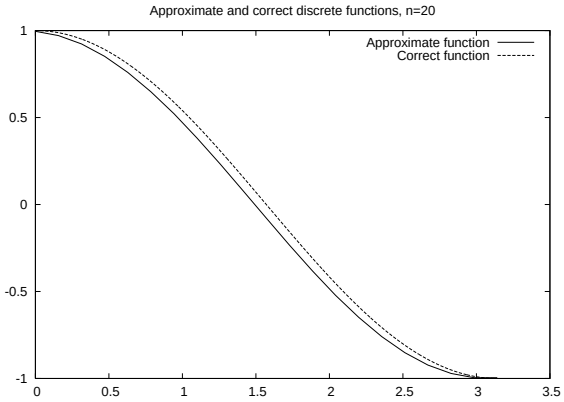
Differences on a Mesh



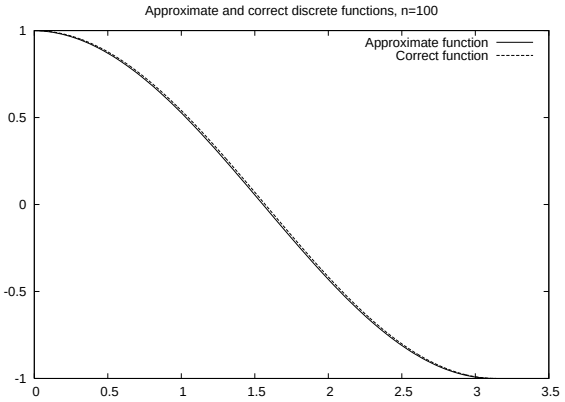
$n = 5$



$n = 10$



$n = 20$



$n = 100$

Plots for exact and approximate derivatives of $\sin(x)$ with varying values of the resolution n.

Taylor Series

$$
\begin{aligned}
f(x_0+h) &\approx f(x_0) \\
f(x_0+h) &\approx f(x_0) + hf'(x_0) \\
f(x_0+h) &\approx f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) \\
f(x_0+h) &\approx f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(x_0) \\
f(x_0+h) &\approx f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(x_0) + \frac{h^4}{24}f''''(x_0) \\
f(x_0+h) &\approx \sum_{k=0}^{n} \frac{h^k}{k!}f^{(k)}(x_0)
\end{aligned}
$$

Derivatives Revisited

We observed aboved that

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

By using the Taylor series, we can obtain this approximation directly, and also get an indication of the error of the approximation.

$$f(x+h) = f(x) + hf'(x) + O(h^2),$$

and thus

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h), \tag{1}$$

More Accurate Difference Approximations

We can also use the Taylor series to derive more accurate approximations of the derivatives.

$$f(x+h) \approx f(x) + hf'(x) + \frac{h^2}{2}f''(x) + O(h^3). \tag{2}$$

By using $-h$ instead of h, we get

$$f(x-h) \approx f(x) - hf'(x) + \frac{h^2}{2}f''(x) + O(h^3). \tag{3}$$

By subtracting (3) from (2), we have

$$f(x+h) - f(x-h) = 2hf'(x) + O(h^3),$$

and consequently

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

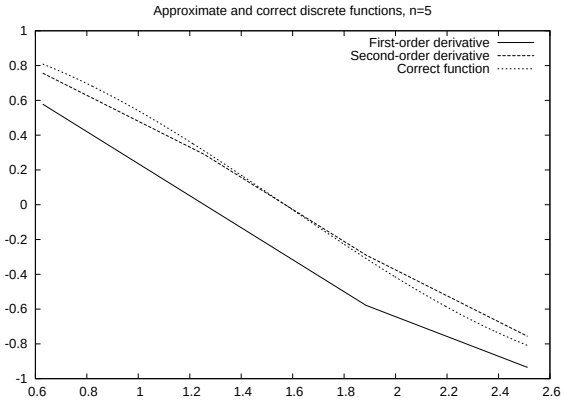
More Accurate Difference Approximations

Note that the error is now $O(h^2)$ whereas the error term of (1) is $O(h)$. In order to see if the error is actually reduced, let us compare the following two approximations

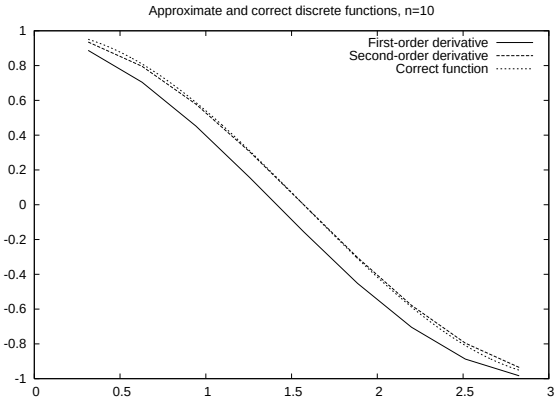
$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \text{ and } f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

by applying them to the discrete version of $\sin(x)$ on the interval $(0, \pi)$.

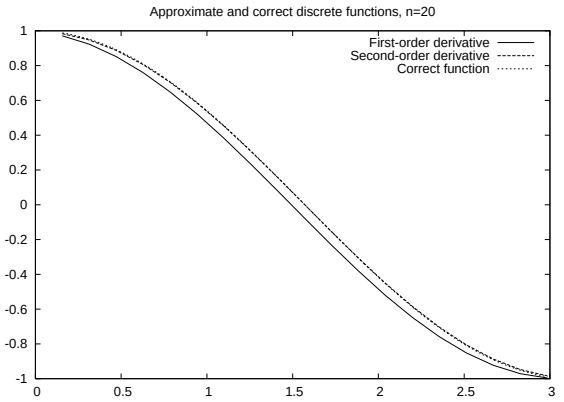
More Accurate Difference Approximations



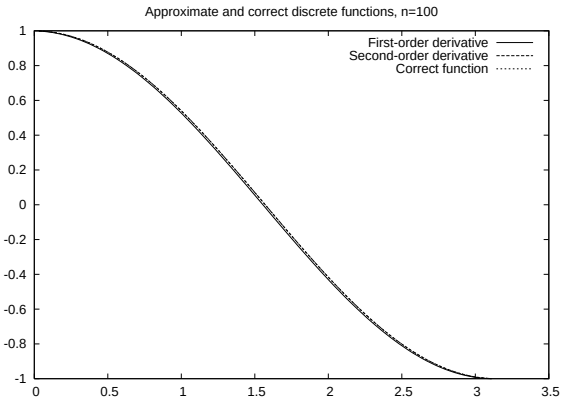
$n = 5$



$n = 10$



$n = 20$



$n = 100$

Plots of exact and approximate derivatives with various number of mesh points n.

Differential Equations

Suppose we want to compute a numerical approximation of the solution of

$$u'(t) = u(t) \tag{5}$$

equipped with the intial condition

$$u(0) = 1. \tag{6}$$

We want to compute approximations from time $t = 0$ to time $t = 1$. Let $n \geqslant 1$ be a given integer, and define

$$\Delta t = 1/n\,. \tag{7}$$

Furthermore, let u_k denote an approximation of $u(t_k)$ where

$$t_k = k\Delta t \tag{8}$$

for $k = 0, 1, \ldots, n$.

Differential Equations

The key step in developing a numerical method for this differential equation is to invoke the Taylor series as applied to the exact solution,

$$u(t_{k+1}) = u(t_k) + \Delta t u'(t_k) + O(\Delta t^2), \tag{9}$$

which implies that

$$u'(t_k) \approx \frac{u(t_{k+1}) - u(t_k)}{\Delta t}. \tag{10}$$

By using (5), we get

$$\frac{u(t_{k+1}) - u(t_k)}{\Delta t} \approx u(t_k). \tag{11}$$

Differential Equations

Recall now that $u(t_k)$ is the exact solution at time t_k, and that u_k is the approximate solution at the same point in time. We now want to determine u_k for all $k \geqslant 0$. Obviously, we start by defining

$$u_0 = u(0) = 1.$$

Since we want $u_k \approx u(t_k)$, we require that u_k satisfy the following equality

$$\frac{u_{k+1} - u_k}{\Delta t} = u_k \tag{12}$$

motivated by (11). It follows that

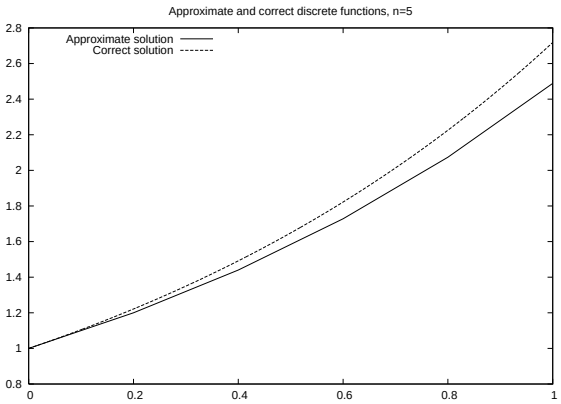
$$u_{k+1} = (1 + \Delta t)u_k \,. \tag{13}$$

Since u_0 is known, we can compute u_1, u_2 and so on by using the formula above.

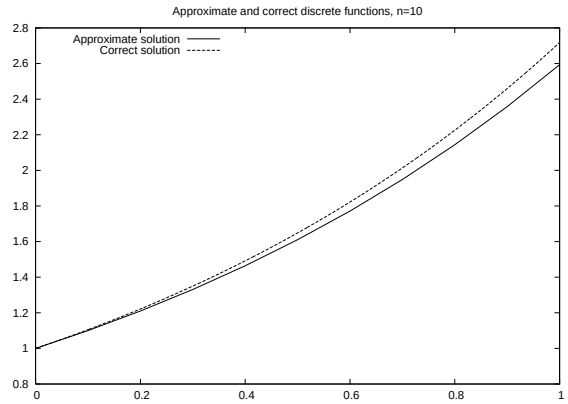
Differential Equations

```
def compute_u(u0, T, n):
    """Solve u'(t)=u(t), u(0)=u0 for t in [0,T] with n steps."
    t = linspace(0, T, n+1)
    t[0] = 0
    u = zeros(n+1)
    u[0] = u0
    dt = T/float(n)
    for k in range(0, n, 1):
        u[k+1] = (1+dt)*u[k]
        t[k+1] = t[k] + dt
    return u, t

from scitools.std import *
n  = int(sys.argv[1])

# special test case: u'(t)=u, u(0)=1, t in [0,1]
T = 1; u0 = 1
u, t = compute_u(u0, T, n)
plot(t, u)
tfine = linspace(0, T, 1001) # for accurate plot
v = exp(tfine)               # correct solution
plot(tfine, v)
```

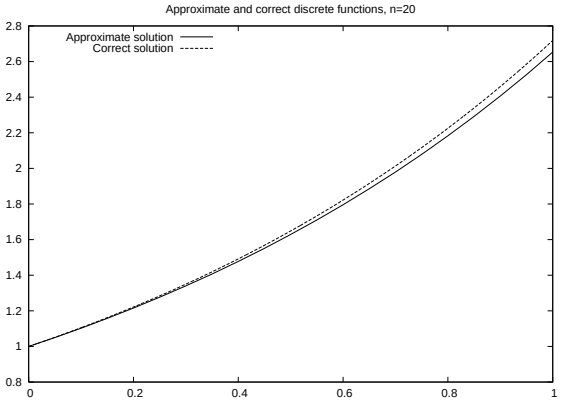
Differential Equations



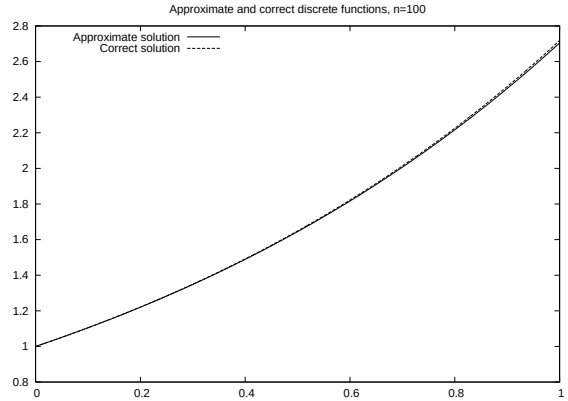
$n = 5$



$n = 10$



$n = 20$



$n = 100$

Plots of exact and approximate solutions of $u'(t) = u(t)$ with varying number of time steps in $[0, 1]$.

A Model for the Spread of a Disease

$I = \textit{Infectives}$ — Have the disease and are able to transmit it.
$S = \textit{Susceptibles}$ — May catch the disease.

A system of differential equations modelling the evolution of S and I is given by

$$S' = -rSI,$$
$$I' = rSI - aI.$$

Here r and a are given constants reflecting the characteristics of the epidemic. The initial conditions are given by

$$S(0) = S_0,$$
$$I(0) = I_0,$$

where the initial state (S_0, I_0) is assumed to be known.

A Model for the Spread of a Disease

Suppose we want to compute numerical solutions of this system from time $t = 0$ to $t = T$. We introduce the time step

$$\Delta t = T/n$$

and the approximations (S_k, I_k) of the solution $(S(t_k), I(t_k))$. An explicit Forward Euler method for the system takes the following form,

$$\frac{S_{k+1} - S_k}{\Delta t} = -rS_k I_k,$$
$$\frac{I_{k+1} - I_k}{\Delta t} = rS_k I_k - aI_k,$$

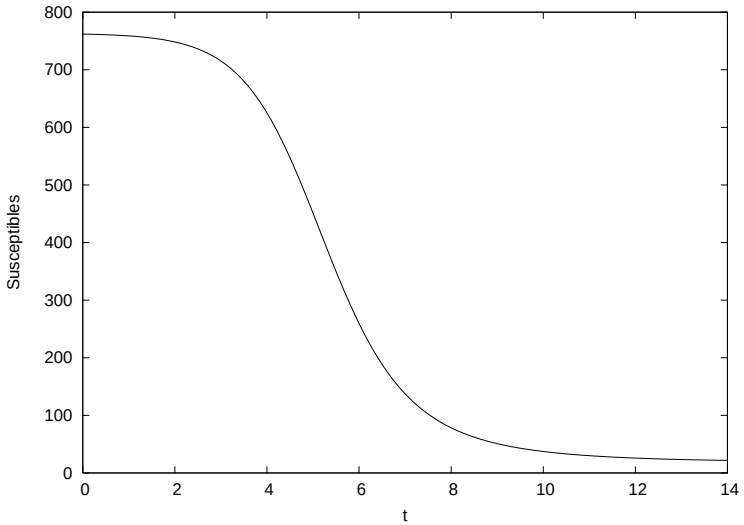
which can be rewritten on computational form

$$S_{k+1} = S_k - \Delta t r S_k I_k,$$
$$I_{k+1} = I_k + \Delta t \left(rS_k I_k - aI_k\right).$$

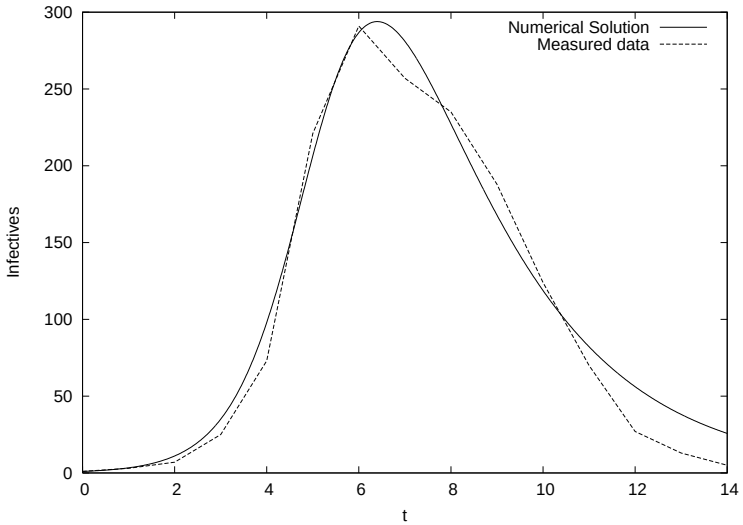
A Model for the Spread of a Disease

We want to apply the program to a specific case where an influenza epidemic hit a British boarding school with a total of 763 boys. The epidemic lasted from 21st January to 4th February in 1978. We let $t = 0$ denote 21st of January and we define $T = 14$ days. We put $S_0 = 762$ and $I_0 = 1$ which means that one person was ill at $t = 0$. In the figure we see the numerical results using $r = 2.18 \times 10^{-3}, a = 0.44, n = 1000$. Also, we have plotted actual the measurements, and we note that the simulations fit the real data quite well.

A Model for the Spread of a Disease



(a)



(b)

Graphs of (a) susceptibles and (b) infectives for an influenza in a British boarding school in 1978.

An Explicit Finite Difference Scheme

The heat equation

$$u_t = u_{xx}\,, \tag{14}$$

defined on the interval $x \in (0,1)$, and for $t > 0$, with boundary conditions

$$u(0,t) = u(1,t) = 0 \tag{15}$$

can be solved analytically for virtually any initial condition

$$u(x,0) = f(x), \tag{16}$$

where $f(x)$ is a given function defined on the interval $(0,1)$.

An Explicit Finite Difference Scheme

Let us start by introducing a computational mesh. Suppose $M > 0$ is a given integer, and define

$$x_i = i/(M+1)$$

for $i = 0, \ldots, M+1$. We note that $x_0 = 0$, and that $x_{M+1} = 1$. These two points define the spatial boundary of the computational mesh, and we know that the solution is zero in these points for all time. Next, we assume that we want to compute the solution for t ranging from $t = 0$ to $t = T$, where $T > 0$ is given. Let $N > 0$ be a given integer, and define a set of discrete points in time:

$$t_n = \frac{n}{N}T, \quad n = 0, \ldots, N\,.$$

Here we note that $t_0 = 0$, and $t_N = T$.

An Explicit Finite Difference Scheme

Our task is to compute an approximation of the exact solution of the heat equation given by the function $u = u(x, t)$. We seek such an approximation in the discrete points given by (x_i, t_n) for $i = 0, \ldots, M + 1$ and $n = 0, \ldots, N$. Let u_i^n denote an approximation of $u(x_i, t_n)$. Then we easily see from the boundary conditions that

$$u_0^n = u_{M+1}^n = 0$$

for $n = 0, \ldots, N$. Thus, at the boundary, we just evaluate the exact boundary conditions so there is no approximation involved. Similarly, at time $t = 0$, we have the initial conditions, and therefore we set

$$u_i^0 = f(x_i)$$

for $i = 1, \ldots M$.

An Explicit Finite Difference Scheme

In order to use the Taylor series in space and time, we need the space-step which is given by

$$\Delta x = x_{i+1} - x_i = \frac{1}{M+1},$$

and the time-step given by

$$\Delta t = t_{n+1} - t_n = \frac{T}{N}\,.$$

The Taylor series in time applied to the analytical solution implies that

$$u_t(x_i, t_n) = \frac{u(x_i, t_{n+1}) - u(x_i, t_n)}{\Delta t} + O(\Delta t),$$

Similarly, we have

$$u_{xx}(x_i, t_n) = \frac{u(x_{i-1}, t_n) - 2u(x_i, t_n) + u(x_{i+1}, t_n)}{\Delta x^2} + O(\Delta x^2).$$

An Explicit Finite Difference Scheme

From the governing partial differential equation it follows that in each computational point we have

$$u_t(x_i, t_n) = u_{xx}(x_i, t_n), \tag{17}$$

and thus

$$\frac{u(x_i, t_{n+1}) - u(x_i, t_n)}{\Delta t} + O(\Delta t) = \frac{u(x_{i-1}, t_n) - 2u(x_i, t_n) + u(x_{i+1}, t_n)}{\Delta x^2} + O(\Delta x^2)$$

Since this holds for the analytical equations, we define our numerical scheme by requiring that the following identity holds

An Explicit Finite Difference Scheme

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2}$$

We can condense the expressions by defining a mesh parameter

$$\rho = \frac{\Delta t}{\Delta x^2}\,.$$

Solving for u_i^{n+1} yields

$$u_i^{n+1} = u_i^n + \rho\left(u_{i-1}^n - 2u_i^n + u_{i+1}^n\right),$$

which can be written even more compactly as

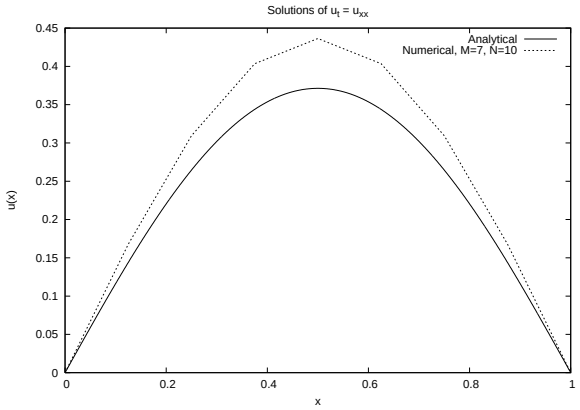
$$u_i^{n+1} = \rho\left(u_{i-1}^n + u_{i+1}^n\right) + (1 - 2\rho)u_i^n. \tag{18}$$

Note that if the numerical solution is known, for all values of i, at time $t = t_n$, then we can compute the solution at time $t = t_{n+1}$ by using this scheme.

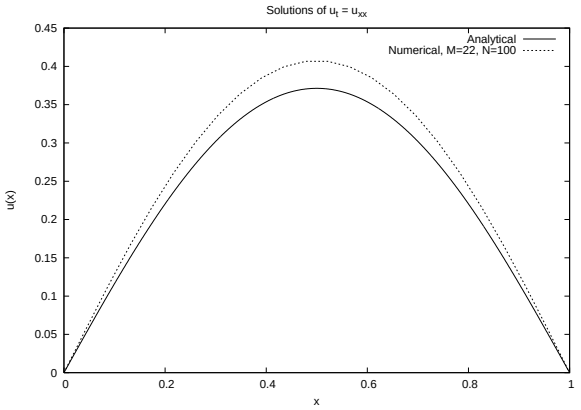
An Explicit Finite Difference Scheme

```
def heat(T, N, M, f):
    # define variables:
    x = linspace(0, 1, M+2)
    t = 0
    dx = 1/float(M+1)
    dt = T/float(N)
    rho = dt/dx**2
    u =  zeros(M+2) # holds u(x_i, t_n)
    um = zeros(M+2) # holds u(x_i, t_n-1)

    # set initial condition:
    for i in range(len(um)):
        um[i] = f(x[i])
    t += dt
    umax = max(um)

    while t <= T:
        # set boundary conditions:
        u[0] = 0; u[-1] = 0
        # use scheme for inner points:
        for i in iseq(1, M):
            u[i] = rho*(um[i-1] + um[i+1]) + (1 - 2*rho)*um[i]

        # plot solution:
        plot(x, u, legend='u(x, t=%g)' % t,
             axis=[0, 1, 0, umax])
        time.sleep(0.5)  # pause, so we can watch the plot
        # prepare for next time step:
        um = u.copy()
        t += dt
```

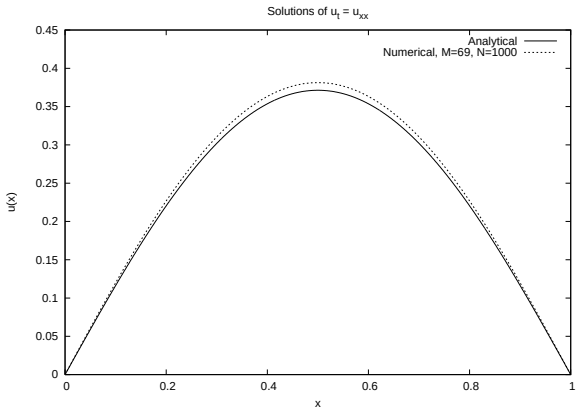
An Explicit Finite Difference Scheme



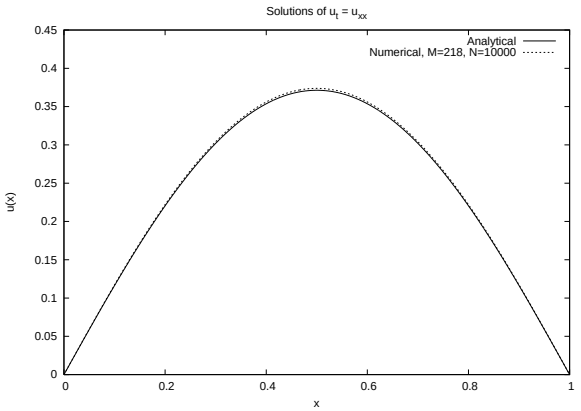
$M = 7$



$M = 22$



$M = 69$



$M = 218$

Plots of the solution $u(x,t)$ of a heat equation for different meshes.

Reaction Diffusion Equations

Equations of the form

$$u_t = u_{xx} + g(u)$$

appear in numerous applications and are called reaction diffusion equations. From a computational point of view, they are more interesting than the heat equation, since, in general, there are no closed form analyitcal solutions available. Even though an analytical solution is impossible, we will see that the equation is surprisingly simple to solve numerically. Suppose, as usual, that we consider this equation on the unit interval with the initial condition given by

$$u(x, 0) = f(x),$$

and that we have standard boundary conditions given by

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

Reaction Diffusion Equations

The time step is defined by

$$\Delta t = \frac{T}{N},$$

and the space-step is

$$\Delta x = \frac{1}{M+1}.$$

The mesh is now defined by

$$x_i = i\Delta x \text{ for } i = 0, \ldots, M+1,$$

and

$$t_n = n\Delta t \text{ for } n = 0, \ldots, N.$$

Reaction Diffusion Equations

As above we let u_i^n denote an approximation of $u(x_i, t_n)$. Recall that

$$u_t(x_i, t_n) = \frac{u(x_i, t_{n+1}) - u(x_i, t_n)}{\Delta t} + O(\Delta t),$$
$$u_{xx}(x_i, t_n) = \frac{u(x_{i-1}, t_n) - 2u(x_i, t_n) + u(x_{i+1}, t_n)}{\Delta x^2} + O(\Delta x^2),$$

and that

$$u_t(x_i, t_n) = u_{xx}(x_i, t_n) + g(u(x_i, t_n)) \tag{19}$$

for $i = 0, \ldots, M+1, n = 0, \ldots, N$. Consequently, we have

$$\frac{u(x_i, t_{n+1}) - u(x_i, t_n)}{\Delta t} + O(\Delta t) = \frac{u(x_{i-1}, t_n) - 2u(x_i, t_n) + u(x_{i+1}, t_n)}{\Delta x^2} + g(u(x_i, t_n)) + O(\Delta x^2).$$

Reaction Diffusion Equations

As above, we define the numerical scheme by requiring that

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2} + g(u_i^n).$$

By recalling that

$$\rho = \frac{\Delta t}{\Delta x^2},$$

we can rewrite the scheme on a computational form,

$$u_i^{n+1} = \rho\left(u_{i-1}^n + u_{i+1}^n\right) + (1 - 2\rho)u_i^n + \Delta t g(u_i^n). \qquad (20)$$

We note again that if the solution is known in all spatial points at time $t = t_n$, we can use the formula above to compute the numerical solution at time $t = t_{n+1}$.

Reaction Diffusion Equations

```
from numpy import *
from scitools.StringFunction import StringFunction
from scitools.numpytools import iseq
import sys

f = sys.argv[1]
g = sys.argv[2]
M = int(sys.argv[3])
N = int(sys.argv[4])
T = eval(sys.argv[5])

f = StringFunction(f)
g = StringFunction(g, independent_variables='u')
x = linspace(0, 1, M + 2)
t = 0
dx = 1/float(M + 1)
dt = T/float(N)
rho = dt/dx**2
print rho
u =  zeros(M + 2) #u(x_i, t_n)
um = zeros(M + 2) #u(x_i, t_n-1)
for i in range(len(um)):
    um[i] = f(x[i])
t += dt

while t <= T:
    u[0] = 0; u[-1] = 0
    for i in range(1, M):
        u[i] = rho*(um[i-1] + um[i+1]) + (1-2*rho)*um[i] + dt*g(um[i])
    um = u.copy()
    t += dt
```

Reaction Diffusion Equations

In the figure you see plots for four numerical solutions of the reaction diffusion equation

$$u_t = u_{xx} + e^u$$

with initial condition

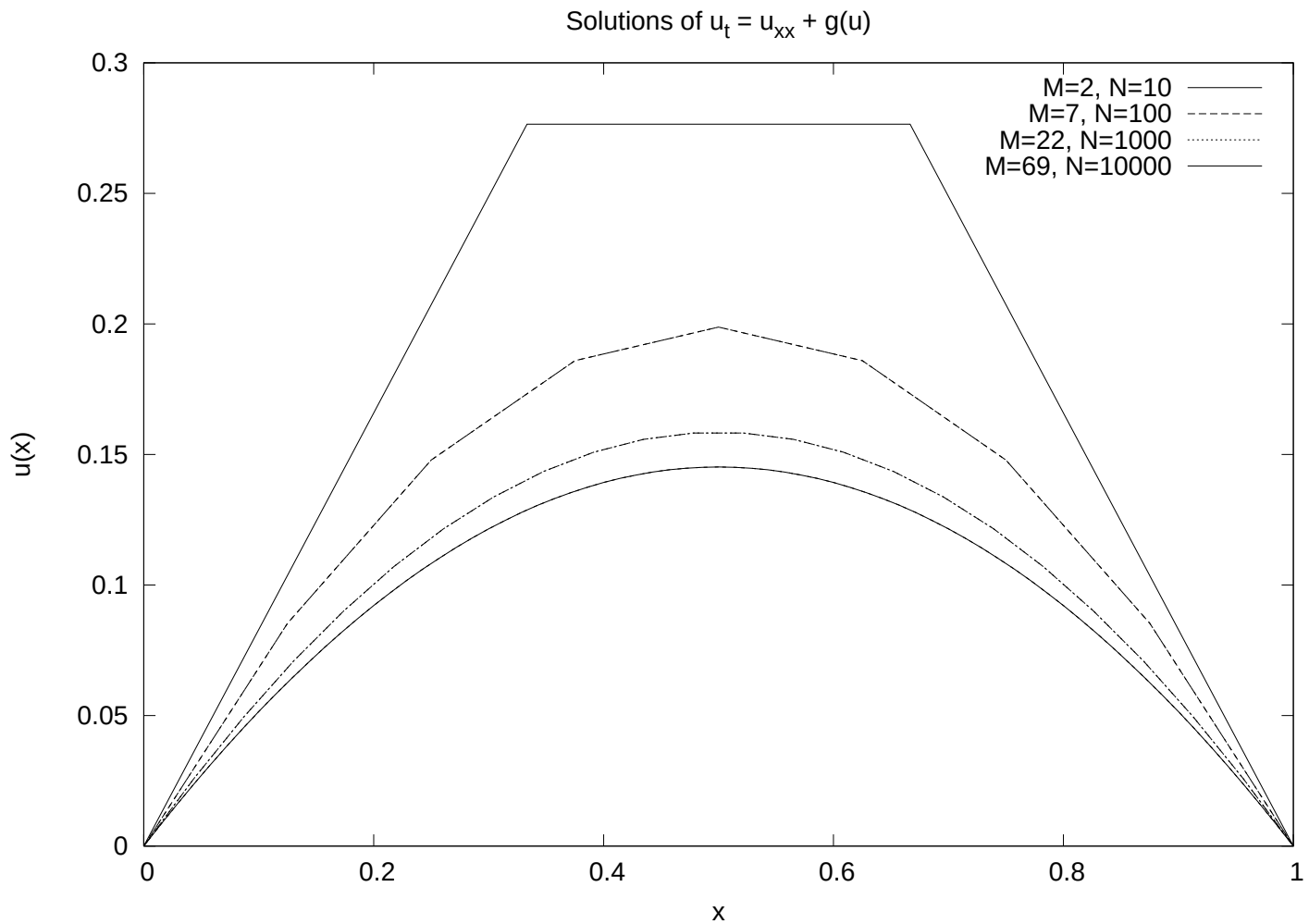
$$u(x, 0) = x(1 - x),$$

and standard boundary conditions given by

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

The plots show the solutions at time $T = 1$ using four different meshes. As usual we observe that the solutions seems to converge to a common limiting function as the mesh is refined.

Reaction Diffusion Equations



Plots of $u_t = u_{xx} + g(u)$ for different meshes.

Chaotic Electrical Waves in the Heart

The following model was introduced by Aliev and Panfilov,

$$e_t = \delta \Delta e - ke(e-a)(e-1) - er,$$
$$r_t = -\left[\epsilon + \frac{\mu_1 r}{\mu_2 + e}\right][r + ke(e-b-1)],$$

where $\delta, a, \epsilon, \mu_1, \mu_2$ and b are given parameters. The variable e models a scaled version of the so called transmembrane potential, whereas r is an auxiliary variable.

Chaotic Electrical Waves in the Heart

We want to solve the system on the unit square, and we start by considering the system equipped with the standard boundary conditions, i.e. we assume that

$$e(x, y, t) = r(x, y, t) = 0$$

for all $(x, y) \in \partial\Omega$. Furthermore, we assume that the inital condition is given by

$$e(x, y, 0) = e^0(x, y),$$
$$r(x, y, 0) = r^0(x, y),$$

Chaotic Electrical Waves in the Heart

$$\frac{e_{i,j}^{n+1} - e_{i,j}^{n}}{\Delta t} = \delta \frac{e_{i+1,j}^{n} - 2e_{i,j}^{n} + e_{i-1,j}^{n}}{h^2} + \delta \frac{e_{i,j+1}^{n} - 2e_{i,j}^{n} + e_{i,j-1}^{n}}{h^2}$$
$$- k e_{i,j}^{n}(e_{i,j}^{n} - a)(e_{i,j}^{n} - 1) - e_{i,j}^{n} r_{i,j}^{n}$$
$$\frac{r_{i,j}^{n+1} - r_{i,j}^{n}}{\Delta t} = -\left(\epsilon + \frac{\mu_1 r_{i,j}^{n}}{\mu_2 + e_{i,j}^{n}}\right)\left(r_{i,j}^{n} + k e_{i,j}^{n}(e_{i,j}^{n} - b - 1)\right)$$

Chaotic Electrical Waves in the Heart

In order to write this scheme on a computational form, we introduce

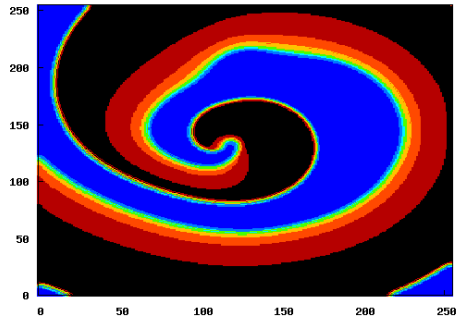
$$\rho = \delta \frac{\Delta t}{h^2},$$

and observe that

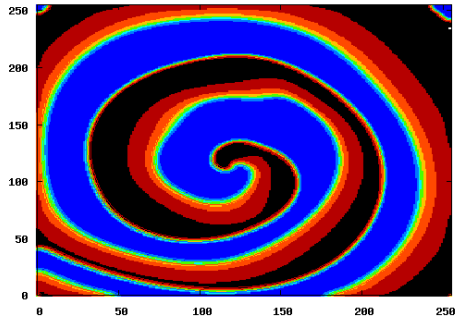
$$e_{i,j}^{n+1} = \rho \left(e_{i+1,j}^n + e_{i-1,j}^n + e_{i,j+1}^n + e_{i,j-1}^n \right) + (1 - 4\rho) e_{i,j}^n - \Delta t \left[k e_{i,j}^n (e_{i,j}^n - a)(e_{i,j}^n - 1) + e_{i,j}^n r_{i,j}^n \right],$$

$$r_{i,j}^{n+1} = r_{i,j}^n - \Delta t \left(\epsilon + \frac{\mu_1 r_{i,j}^n}{\mu_2 + e_{i,j}^n} \right) \left(r_{i,j}^n + k e_{i,j}^n (e_{i,j}^n - b - 1) \right).$$

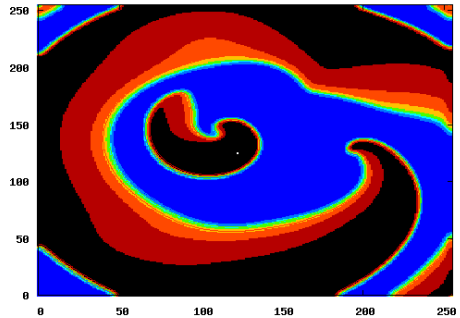
Chaotic Electrical Waves in the Heart



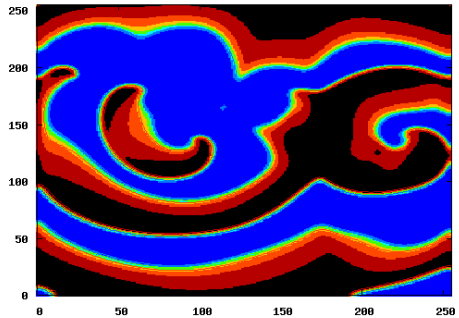
$t = 300$



$t = 400$



$t = 500$



$t = 600$

Chaotic electrical waves in the heart for different t.