Differential equations using Scilab

Kannan M. Moudgalya
IIT Bombay
http://moudgalya.org
kannan@iitb.ac.in

Scilab Training
MITCOE
21 December 2009

Outline

- **Weight reduction ODE model - analytical solution**
- **Numerical integration**
  - **Functions in Scilab**
  - **Euler's method**
- **Predator-prey system**
  - **Modelling**
  - **Euler method - user created integrator**
  - **Backward difference method - built-in function**

Weight Reduction Model

- **Weight of person = x kg**

Weight Reduction Model

- **Weight of person = x kg**
- **Tries to reduce weight**

Weight Reduction Model

- **Weight of person** $= x$ **kg**
- **Tries to reduce weight**
- **Weight loss per month** $= 10\%$ **of weight**

Weight Reduction Model

- **Weight of person =** x **kg**
- **Tries to reduce weight**
- **Weight loss per month =** 10% **of weight**
- **Starting weight = 100 kg**

Weight Reduction Model

- **Weight of person =** x **kg**
- **Tries to reduce weight**
- **Weight loss per month =** 10% **of weight**
- **Starting weight = 100 kg**

$$\frac{dx}{dt} = -0.1x$$

Weight Reduction Model

- **Weight of person =** x **kg**
- **Tries to reduce weight**
- **Weight loss per month =** 10% **of weight**
- **Starting weight = 100 kg**

$$\frac{dx}{dt} = -0.1x$$

Initial conditions:

$$x = 100$$

Weight Reduction Model

- **Weight of person =** x **kg**
- **Tries to reduce weight**
- **Weight loss per month =** 10% **of weight**
- **Starting weight = 100 kg**

$$\frac{dx}{dt} = -0.1x$$

Initial conditions:

$$x = 100 \quad \textbf{at t=0}$$

Weight Reduction Model

- **Weight of person =** x **kg**
- **Tries to reduce weight**
- **Weight loss per month =** 10% **of weight**
- **Starting weight = 100 kg**

$$\frac{dx}{dt} = -0.1x$$

Initial conditions:

$$x = 100 \quad \textbf{at t=0}$$

Determine $x(t)$ **as a function of** t**.**

Analytical Solution of Simple Model

Recall the model:

$$\frac{dx}{dt} = -0.1x$$

$$x(t = 0) = 100$$

Cross multiplying, $$\frac{dx}{x} = -0.1dt$$

Integrating both sides from 0 to t**,**

$$\int \frac{dx}{x} = -0.1 \int dt$$

$$C + \ln x(t) = -0.1t$$

Using initial conditions,

$$C = -\ln 100$$

Thus, the final solution is,

$$\ln \frac{x(t)}{100} = -0.1t \quad \textbf{or} \quad x(t) = 100e^{-0.1t}$$

Summary of Weight Reduction Problem

- **Weight of person = x kg**
- **Tries to reduce weight**
- **Weight loss per month = 10% of weight**
- **Starting weight = 100 kg**

$$x(t) = 100e^{-0.1t}$$

Compute and plot for two years, i.e. for 24 months:

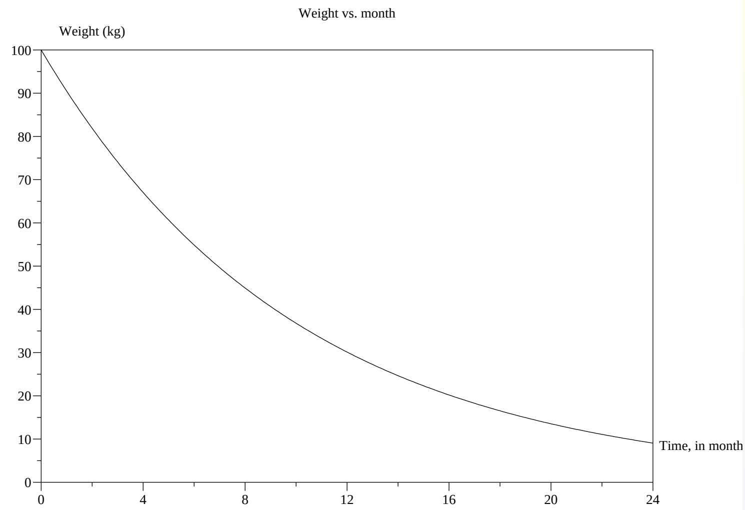
Summary of Weight Reduction Problem

- **Weight of person = x kg**
- **Tries to reduce weight**
- **Weight loss per month = 10% of weight**
- **Starting weight = 100 kg**

$$x(t) = 100e^{-0.1t}$$

Compute and plot for two years, i.e. for 24 months:

```
T=0:0.1:24;
plot2d(T,100*exp(-0.1*T));
xtitle('Weight vs. month','Time in months',
        'Weight (kg)')
```


Weight vs. month
Weight (kg)
100
90
80
70
60
50
40
30
20
10
0
0
4
8
12
16
20
24
Time, in month

Need for Numerical Solution

- **Exact solution is ok for simple models**

Need for Numerical Solution

- **Exact solution is ok for simple models**
- **What if the model is complicated?**

Need for Numerical Solution

- **Exact solution is ok for simple models**
- **What if the model is complicated?**
- **Consider integrating the more difficult problem:**

$$\frac{dx}{dt} = \frac{2 + 18t + 68t^2 + 180t^3 + 250t^4 + 250t^5}{x^2}$$

with initial condition,

$$x(t = 0) = 1$$

Need for Numerical Solution

- **Exact solution is ok for simple models**
- **What if the model is complicated?**
- **Consider integrating the more difficult problem:**

$$\frac{dx}{dt} = \frac{2 + 18t + 68t^2 + 180t^3 + 250t^4 + 250t^5}{x^2}$$

with initial condition,

$$x(t = 0) = 1$$

- **Analytical (i.e. exact) solution difficult to find**

Simplest Numerical Solution: Explicit Euler

- **Suppose that we want to integrate the following system:**

$$\frac{dx}{dt} = g(x, t)$$

with initial condition:

$$x(t = 0) = x_0$$

Simplest Numerical Solution: Explicit Euler

- **Suppose that we want to integrate the following system:**

$$\frac{dx}{dt} = g(x, t)$$

with initial condition:

$$x(t = 0) = x_0$$

- **Approximate numerical method - divide time into equal intervals:** t_0**,** t_1**,** t_2**, etc.**

$$\frac{x_n - x_{n-1}}{\Delta t} = g(x_{n-1}, t_{n-1})$$

Simplest Numerical Solution: Explicit Euler

- **Suppose that we want to integrate the following system:**

$$\frac{dx}{dt} = g(x, t)$$

with initial condition:

$$x(t = 0) = x_0$$

- **Approximate numerical method - divide time into equal intervals:** t_0, t_1, t_2, **etc.**

$$\frac{x_n - x_{n-1}}{\Delta t} = g(x_{n-1}, t_{n-1})$$

- **Simplifying,**

$$x_n - x_{n-1} = \Delta t \, g(x_{n-1}, t_{n-1})$$
$$x_n = x_{n-1} + \Delta t \, g(x_{n-1}, t_{n-1})$$

Simplest Numerical Solution: Explicit Euler

- **Suppose that we want to integrate the following system:**

$$\frac{dx}{dt} = g(x, t)$$

with initial condition:

$$x(t = 0) = x_0$$

- **Approximate numerical method - divide time into equal intervals:** t_0, t_1, t_2, **etc.**

$$\frac{x_n - x_{n-1}}{\Delta t} = g(x_{n-1}, t_{n-1})$$

- **Simplifying,**

$$x_n - x_{n-1} = \Delta t \, g(x_{n-1}, t_{n-1})$$
$$x_n = x_{n-1} + \Delta t \, g(x_{n-1}, t_{n-1})$$

- **Given** x_0**, can march forward and determine** x_n **for all future** n**.**

Example revisited

Recall the problem statement for numerical solution:

$$\frac{dx}{dt} = \frac{2 + 18t + 68t^2 + 180t^3 + 250t^4 + 250t^5}{x^2}$$

with initial condition,

$$x(t = 0) = 1$$

Recall the Euler method:

$$\frac{dx}{dt} = g(x, t)$$

Solution for initial condition, $x(t = 0) = x_0$ **is,**

$$x_n = x_{n-1} + \Delta t \, g(x_{n-1}, t_{n-1})$$

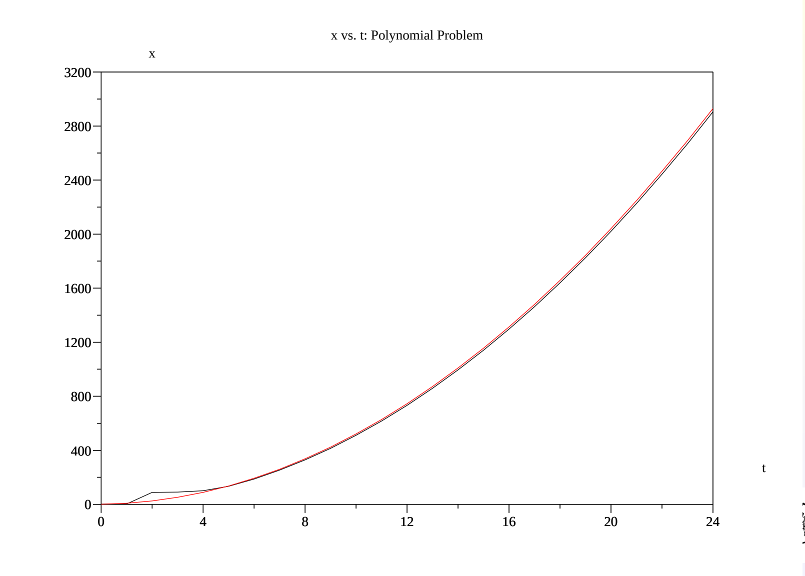
Scilab Code

```
getf("diff1.sci");
getf("Euler.sci");
x0=1; t0=0; T=0:0.1:24;
sol = Euler(x0,t0,T,diff1);
// sol = ode(x0,t0,T,diff1);
plot2d(T,sol), pause
plot2d(T,1+2*T+5*T^2,5)
xtitle('x vs. t: Polynomial Problem','t','x')
```

```
function x = Euler(x0,t0,t,g)
n = length(t), x = x0;
for j = 1:n-1
    x0 = x0 + (t(j+1)-t(j))*g(t(j),x0);
    x = [x x0];
end;
```

```
function xdot = diff1(t,x)
xdot = (2+18*t+68*t^2+180*t^3+250*t^4+250*t^5)/x^2;
```

Numerical Solution, Compared with Exact Solution



Predator-Prey Problem

- **Population dynamics of predator-prey**

Predator-Prey Problem

- **Population dynamics of predator-prey**
- **Prey can find food, but gets killed on meeting predator**

Predator-Prey Problem

- **Population dynamics of predator-prey**
- **Prey can find food, but gets killed on meeting predator**
- **Examples: parasites and certain hosts; wolves and rabbits**

Predator-Prey Problem

- **Population dynamics of predator-prey**
- **Prey can find food, but gets killed on meeting predator**
- **Examples: parasites and certain hosts; wolves and rabbits**
- $x_1(t)$ **- number of prey;** $x_2(t)$ **- number of predator at time t**

Predator-Prey Problem

- **Population dynamics of predator-prey**
- **Prey can find food, but gets killed on meeting predator**
- **Examples: parasites and certain hosts; wolves and rabbits**
- $x_1(t)$ **- number of prey;** $x_2(t)$ **- number of predator at time t**
- **Prey, if left alone, grows at a rate proportional to** x_1

Predator-Prey Problem

- **Population dynamics of predator-prey**
- **Prey can find food, but gets killed on meeting predator**
- **Examples: parasites and certain hosts; wolves and rabbits**
- $x_1(t)$ **- number of prey;** $x_2(t)$ **- number of predator at time t**
- **Prey, if left alone, grows at a rate proportional to** x_1
- **Predator, on meeting prey, kills it** $\Rightarrow$ **proportional to** $x_1 x_2$

$$\frac{dx_1}{dt} = 0.25x_1 - 0.01x_1x_2$$

Predator-Prey Problem

- **Population dynamics of predator-prey**
- **Prey can find food, but gets killed on meeting predator**
- **Examples: parasites and certain hosts; wolves and rabbits**
- $x_1(t)$ **- number of prey;** $x_2(t)$ **- number of predator at time t**
- **Prey, if left alone, grows at a rate proportional to** x_1
- **Predator, on meeting prey, kills it** $\Rightarrow$ **proportional to** $x_1 x_2$

$$\frac{dx_1}{dt} = 0.25x_1 - 0.01x_1x_2$$

- **Predator, if left alone, decrease by natural causes**

Predator-Prey Problem

- Population dynamics of predator-prey
- Prey can find food, but gets killed on meeting predator
- Examples: parasites and certain hosts; wolves and rabbits
- $x_1(t)$ - number of prey; $x_2(t)$ - number of predator at time t
- Prey, if left alone, grows at a rate proportional to x_1
- Predator, on meeting prey, kills it $\Rightarrow$ proportional to $x_1 x_2$

$$\frac{dx_1}{dt} = 0.25x_1 - 0.01x_1x_2$$

- Predator, if left alone, decrease by natural causes
- Predators increase their number on meeting prey

$$\frac{dx_2}{dt} = -x_2 + 0.01x_1x_2$$

Predator-Prey Problem

- **Population dynamics of predator-prey**
- **Prey can find food, but gets killed on meeting predator**
- **Examples: parasites and certain hosts; wolves and rabbits**
- $x_1(t)$ **- number of prey;** $x_2(t)$ **- number of predator at time t**
- **Prey, if left alone, grows at a rate proportional to** x_1
- **Predator, on meeting prey, kills it** $\Rightarrow$ **proportional to** $x_1 x_2$

$$\frac{dx_1}{dt} = 0.25x_1 - 0.01x_1x_2$$

- **Predator, if left alone, decrease by natural causes**
- **Predators increase their number on meeting prey**

$$\frac{dx_2}{dt} = -x_2 + 0.01x_1x_2$$

- **Determine** $x_1(t)$, $x_2(t)$ **when** $x_1(0) = 80$, $x_2(0) = 30$

Explicit Euler for a System of Equations

$$\frac{dx_1}{dt} = g_1(x_1, \ldots, x_n, t)$$

$$\vdots$$

$$\frac{dx_N}{dt} = g_n(x_1, \ldots, x_n, t)$$

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} g_1(x_1, \ldots, x_n, t-1) \\ \vdots \\ g_n(x_1, \ldots, x_n, t-1) \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}_t = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}_{t-1} + \Delta t \begin{bmatrix} g_1((x_1, \ldots, x_n)|_{t-1}, t-1) \\ \vdots \\ g_N((x_1, \ldots, x_n)|_{t-1}, t-1) \end{bmatrix}$$

Solution in vector form:

$$\underline{x}_t = \underline{x}_{t-1} + \Delta t \underline{g}(\underline{x}_{t-1})$$

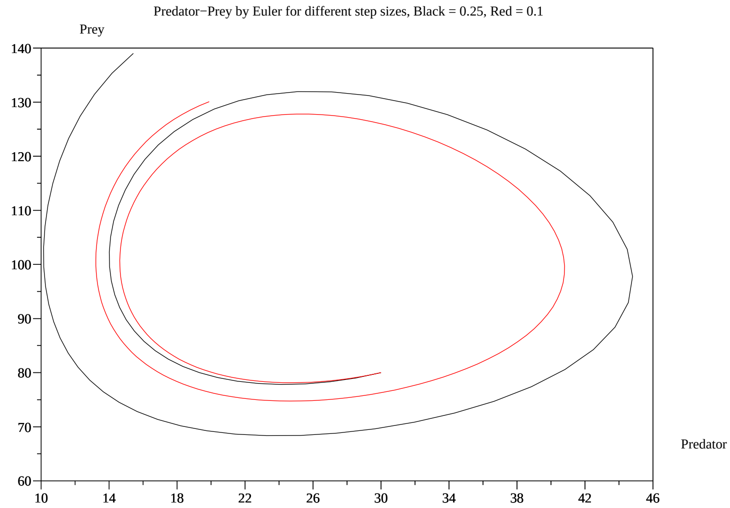
Scilab Code for Predator-Prey Problem

```
getf("pred.sci");
getf("Euler.sci");
x0=[80,30]'; t0=0; T=0:0.1:20; T=T';
// sol = Euler(x0,t0,T,pred);
sol = ode(x0,t0,T,pred);
clf();
plot2d(T,sol')
xset('window',1)
plot2d(sol(2,:),sol(1,:))
```

```
function x = Euler(x0,t0,t,g)
n = length(t), x = x0;
for j = 1:n-1
      x0 = x0 + (t(j+1)-t(j))*g(t(j),x0);
      x = [x x0];
end;
```

```
function xdot = pred(t,x)
xdot(1) = 0.25*x(1)-0.01*x(1)*x(2);
xdot(2) = -x(2)+0.01*x(1)*x(2);
```

Predator-Prey Problem: Solution by Euler



As step size increases, the solution diverges more from the actual

General method to handle stiff systems

- **The predator-prey problem is an example of a stiff system**

General method to handle stiff systems

- **The predator-prey problem is an example of a stiff system**
- **Results because of sudden changes in the derivative**

General method to handle stiff systems

- **The predator-prey problem is an example of a stiff system**
- **Results because of sudden changes in the derivative**
- **Approximation of using previous time values does not work**

General method to handle stiff systems

- The predator-prey problem is an example of a stiff system
- Results because of sudden changes in the derivative
- Approximation of using previous time values does not work
- General approach to solve this problem:

$$x_n = x_{n-1} + \Delta t \, g(x_n, t_n)$$

General method to handle stiff systems

- **The predator-prey problem is an example of a stiff system**
- **Results because of sudden changes in the derivative**
- **Approximation of using previous time values does not work**
- **General approach to solve this problem:**

$$x_n = x_{n-1} + \Delta t \, g(x_n, t_n)$$

- **Requires solution by trial and error, as $g(x_n, t_n)$ is unknown**

General method to handle stiff systems

- **The predator-prey problem is an example of a stiff system**
- **Results because of sudden changes in the derivative**
- **Approximation of using previous time values does not work**
- **General approach to solve this problem:**

$$x_n = x_{n-1} + \Delta t \, g(x_n, t_n)$$

- **Requires solution by trial and error, as $g(x_n, t_n)$ is unknown**
- **Scilab has state of the art methods (`ode`) to solve such systems**

General method to handle stiff systems

- **The predator-prey problem is an example of a stiff system**
- **Results because of sudden changes in the derivative**
- **Approximation of using previous time values does not work**
- **General approach to solve this problem:**

$$x_n = x_{n-1} + \Delta t \, g(x_n, t_n)$$

- **Requires solution by trial and error, as $g(x_n, t_n)$ is unknown**
- **Scilab has state of the art methods (`ode`) to solve such systems**
- **Derived from ODEPACK**

General method to handle stiff systems

- **The predator-prey problem is an example of a stiff system**
- **Results because of sudden changes in the derivative**
- **Approximation of using previous time values does not work**
- **General approach to solve this problem:**

$$x_n = x_{n-1} + \Delta t \, g(x_n, t_n)$$

- **Requires solution by trial and error, as $g(x_n, t_n)$ is unknown**
- **Scilab has state of the art methods (`ode`) to solve such systems**
- **Derived from ODEPACK**
  - **FOSS**

General method to handle stiff systems

- **The predator-prey problem is an example of a stiff system**
- **Results because of sudden changes in the derivative**
- **Approximation of using previous time values does not work**
- **General approach to solve this problem:**

$$x_n = x_{n-1} + \Delta t \, g(x_n, t_n)$$

- **Requires solution by trial and error, as $g(x_n, t_n)$ is unknown**
- **Scilab has state of the art methods (`ode`) to solve such systems**
- **Derived from ODEPACK**
  - **FOSS**
  - **In use for thirty years**

General method to handle stiff systems

- **The predator-prey problem is an example of a stiff system**
- **Results because of sudden changes in the derivative**
- **Approximation of using previous time values does not work**
- **General approach to solve this problem:**

$$x_n = x_{n-1} + \Delta t \, g(x_n, t_n)$$

- **Requires solution by trial and error, as $g(x_n, t_n)$ is unknown**
- **Scilab has state of the art methods (`ode`) to solve such systems**
- **Derived from ODEPACK**
  - **FOSS**
  - **In use for thirty years**
  - **Bugs have been removed by millions of users**

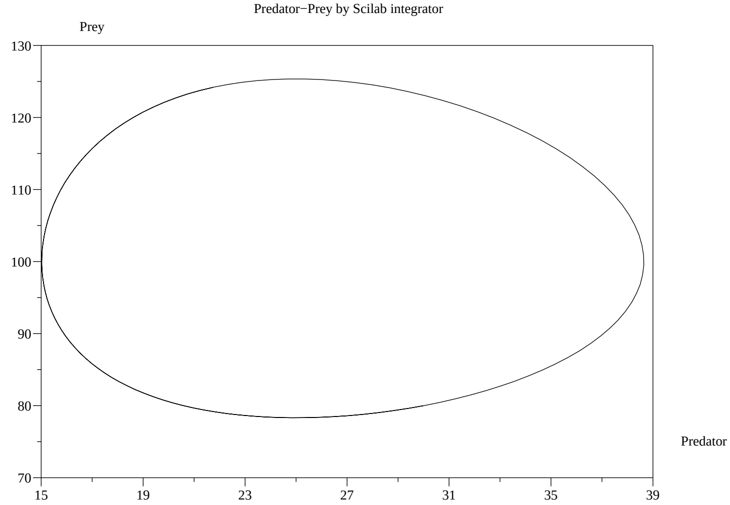
Predator-Prey Problem by Scilab Integrator

Execute the following code, after commenting out `Euler` and uncommenting `ode`:

```
getf("pred.sci");
getf("Euler.sci");
x0=[80,30]'; t0=0; T=0:0.1:20; T=T';
// sol = Euler(x0,t0,T,pred);
sol = ode(x0,t0,T,pred);
clf();
plot2d(T,sol')
xset('window',1)
plot2d(sol(2,:),sol(1,:))
```

```
function xdot = pred(t,x)
xdot(1) = 0.25*x(1)-0.01*x(1)*x(2);
xdot(2) = -x(2)+0.01*x(1)*x(2);
```

Predator-Prey Problem: Solution by Scilab Integrator



Use the Scilab built-in integrator to get the correct solution

Partial Differential Equations

Parabolic Differential Equations

- **Heat conduction equation**

Parabolic Differential Equations

- **Heat conduction equation**
- **Diffusion equation**

$$\frac{\partial u(t,x)}{\partial t} = c\frac{\partial^2 u(t,x)}{\partial x^2}$$

Parabolic Differential Equations

- **Heat conduction equation**
- **Diffusion equation**

$$\frac{\partial u(t,x)}{\partial t} = c\frac{\partial^2 u(t,x)}{\partial x^2}$$

- **Initial condition:**

$$u(0,x) = g(x), \quad 0 \le x \le 1$$

Parabolic Differential Equations

- **Heat conduction equation**
- **Diffusion equation**

$$\frac{\partial u(t,x)}{\partial t} = c\frac{\partial^2 u(t,x)}{\partial x^2}$$

- **Initial condition:**

$$u(0,x) = g(x), \quad 0 \le x \le 1$$

- **Boundary conditions:**

$$u(t,0) = \alpha, \quad u(t,1) = \beta, \quad t \ge 0$$

Parabolic Differential Equations

- **Heat conduction equation**
- **Diffusion equation**

$$\frac{\partial u(t,x)}{\partial t} = c\frac{\partial^2 u(t,x)}{\partial x^2}$$

- **Initial condition:**

$$u(0,x) = g(x), \quad 0 \le x \le 1$$

- **Boundary conditions:**

$$u(t,0) = \alpha, \quad u(t,1) = \beta, \quad t \ge 0$$

- **Let u_j^m be approximate solution at $x_j = j\Delta x$, $t_m = m\Delta t$**

$$\frac{u_j^{m+1} - u_j^m}{\Delta t} = \frac{c}{(\Delta x)^2}(u_{j-1}^m - 2u_j^m + u_{j+1}^m)$$

Finite Difference Approach

$$\frac{u_j^{m+1} - u_j^m}{\Delta t} = \frac{c}{(\Delta x)^2}(u_{j-1}^m - 2u_j^m + u_{j+1}^m), \quad \mu = \frac{c\Delta t}{(\Delta x)^2}$$
$$u_j^{m+1} = u_j^m + \mu(u_{j-1}^m - 2u_j^m + u_{j+1}^m)$$
$$= \mu u_{j-1}^m + (1-2\mu)u_j^m + \mu u_{j+1}^m$$

Write this equation at every spatial grid:

$$u_1^{m+1} = \mu u_0^m + (1-2\mu)u_1^m + \mu u_2^m$$
$$u_2^{m+1} = \mu u_1^m + (1-2\mu)u_2^m + \mu u_3^m$$
$$\vdots$$
$$u_N^{m+1} = \mu u_{N-1}^m + (1-2\mu)u_N^m + \mu u_{N+1}^m$$

Finite Difference Approach - Continued

$$
\begin{aligned}
u_1^{m+1} &= \mu u_0^m + (1-2\mu)u_1^m + \mu u_2^m \\
u_2^{m+1} &= \mu u_1^m + (1-2\mu)u_2^m + \mu u_3^m \\
&\vdots \\
u_N^{m+1} &= \mu u_{N-1}^m + (1-2\mu)u_N^m + \mu u_{N+1}^m
\end{aligned}
$$

In matrix form,

$$
\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{bmatrix}^{m+1} = \begin{bmatrix} 1-2\mu & \mu & & & \\ \mu & 1-2\mu & \mu & & \\ & & \ddots & & \\ & & & \mu & 1-2\mu \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{bmatrix}^{m} + \begin{bmatrix} \mu u_0^m \\ 0 \\ \vdots \\ \mu u_{N+1}^m \end{bmatrix}
$$

Conclusions

- **Scilab is ideal for educational institutions, including schools**

Conclusions

- **Scilab is ideal for educational institutions, including schools**
- **Built on a sound numerical platform**

Conclusions

- **Scilab is ideal for educational institutions, including schools**
- **Built on a sound numerical platform**
- **It has good integrators for differential equations**

Conclusions

- Scilab is ideal for educational institutions, including schools
- Built on a sound numerical platform
- It has good integrators for differential equations
- It is free

Conclusions

- **Scilab is ideal for educational institutions, including schools**
- **Built on a sound numerical platform**
- **It has good integrators for differential equations**
- **It is free**
- **Also suitable for industrial applications**

Thank you