Inf165 Chapter 2

18th September 2003

Exercises

2.1

a) $r'(t) = e^{4t} = 1 + 4r(t),\ r(0) = \frac{1}{4}(e^0 - 1) = 0$

b) Use the approximation $y'(t_n) \approx \frac{y(t_{n+1}) - y(t_n)}{\Delta t}$.

$$\frac{y_{n+1} - y_n}{\Delta t} = 1 + 4y_n \Rightarrow y_{n+1} = y_n + \Delta t(1 + 4y_n)$$

c) Use the approximation $z'(t_{n+1}) \approx \frac{z(t_{n+1}) - z(t_n)}{\Delta t}$.

$$\frac{z_{n+1} - z_n}{\Delta t} = 1 + 4z_{n+1} \Rightarrow z_{n+1} = \frac{z_n + \Delta t}{1 - 4\Delta t}$$

d)

$y_1 = \frac{1}{10}$	$z_1 = \frac{1}{6}$	$r(\Delta t) = 0.12$
$y_2 = \frac{24}{100}$	$z_2 = \frac{4}{9}$	$r(2\Delta t) = 0.30$
$y_3 = \frac{109}{250}$	$z_3 = \frac{49}{54}$	$r(3\Delta t) = 0.58$

e) See the program **ex21.m**.

f) $E/\Delta t \approx 109$.

2.2

a) $r'(t) = e^t + 2t = r(t) + 2t - t^2,\ r(0) = e^0 + 0^2 = 1$

b) Use the approximation $y'(t_n) \approx \frac{y(t_{n+1}) - y(t_n)}{\Delta t}$.

$$\frac{y_{n+1} - y_n}{\Delta t} = y_n + 2t_n - t_n^2 \Rightarrow y_{n+1} = y_n + \Delta t(y_n - 2t_n - t_n^2)$$

c) Use the approximation $z'(t_{n+1}) \approx \frac{z(t_{n+1}) - z(t_n)}{\Delta t}$.

$$\frac{z_{n+1} - z_n}{\Delta t} = y_{n+1} + 2t_{n+1} - t_{n+1}^2 \Rightarrow z_{n+1} = \frac{z_n + \Delta t(2t_{n+1} - t_{n+1}^2)}{1 + \Delta t}$$

d)

$y_1 = 1.1$	$z_1 = 1.13$	$r(\Delta t) = 1.115$
$y_2 = 1.229$	$z_2 = 1.296$	$r(2\Delta t) = 1.269$
$y_3 = 1.388$	$z_3 = 1.497$	$r(3\Delta t) = 1.440$

e) See the program **ex22.m**.

f) $E/\Delta t \approx 3.1$

2.3

a) $r(1) = r_0 e^{50 \cdot 1}$ $(= 5.18 \cdot 10^{24})$

b) Upper estimate is $1100e^{50}$ $(= 5.70 \cdot 10^{24})$.
Lower estimate is $900e^{50}$ $(= 4.67 \cdot 10^{24})$.

c) Upper estimate is $1100e^{60}$ $(= 1.26 \cdot 10^{29})$.
Lower estimate is $900e^{40}$ $(= 2.12 \cdot 10^{20})$.

2.4

a) $r(1) = 8790$ $(= R)$

b) Upper *and* lower estimate is 8790.

c) Upper *and* lower estimate is 8790.

d) Upper estimate is 10000, lower estimate is 7000.

2.5

a) $a = 0.03, r_0 = 5.3, t =$years since 1800; $r(t) = 5.3e^{0.03t}$.

b) When defining t as the year the formula becomes $r(t) = 5.3e^{0.03(t-1800)}$.

c) See the program **ex25.m**.

d) $r(1980) = 1173$.

e) See the program **ex25.m**. 290 is a good value for R. (Try it!) This gives the result $r(1980) = 233.4$.

2.6

a)

$$r_n = 1 \Rightarrow\ r_{n+1} = \frac{1}{2\Delta t}\left(-1 + \Delta t + \sqrt{1 + 2\Delta t + (\Delta t)^2}\right) = 1$$

If r_n is 1 then $r_{n+1} = 1$. By simple induction we find that if $r_0 = 1$ then $r_n = 1$ for all $n \geq 0$.

$$r_n = 0 \Rightarrow\ r_{n+1} = \frac{1}{2\Delta t}\left(-1 + \Delta t + \sqrt{(1 - \Delta t)^2}\right) = 0$$

If r_n is 0 then $r_{n+1} = 0$. By simple induction we find that if $r_0 = 0$ then $r_n = 0$ for all $n \geq 0$.

b) $f(1) = 1 - r_n$ We know that $(1 - r_n) > 0$ since $r_n > 1$.
$f(r_n) = -\Delta t r_n(1 - r_n)$ This must be negative, as we know that r_n, $(r_n - 1)$, and Δt, are all positive.

c) $f(0) = -r_n$ We know that $r_n > 0$.

$$f(-\infty) = \lim_{r \to -\infty} (1 - \Delta t)r + \Delta t r^2 - r_n > 0$$

e) For $r_n > 1$:
$f(r_n) = -\Delta t r_n(1 - r_n) > 0$
$f(1) = 1 - r_n < 0$

2.7

a)

$$u(t_{n+1}) = \int_{-\infty}^{t_{n+1}} u'(t)\,dt = \int_{-\infty}^{t_n} u'(t)\,dt + \int_{t_n}^{t_{n+1}} u'(t)\,dt = u(t_n) + \int_{t_n}^{t_{n+1}} f\,(u(t))\;dt$$

b) Trapezoid: Replace $\boxed{\int_{t_n}^{t_{n+1}} f(u(t))\,dt}$ with $\boxed{\frac{\Delta t}{2}\left(f(u_{n+1}) + f(u_n)\right)}$ in (2.58).

c) Midpoint: Replace $\boxed{\int_{t_n}^{t_{n+1}} f(u(t))\,dt}$ with $\boxed{\Delta t f\left(\frac{1}{2}(u_{n+1} + u_n)\right)}$ in (2.58).

d) Simpsons: Replace $\boxed{\int_{t_n}^{t_{n+1}} f(u(t))\,dt}$ with $\boxed{\frac{\Delta t}{6}\left(f(u_{n+1}) + 4f\left(\frac{1}{2}(u_{n+1} + u_n)\right) + f(u_n)\right)}$ in (2.58).

e) See the program **ex27.java**.

2.8

a) $y(t) = e^t$

b) $y(t) = \frac{\alpha}{1 - t\alpha}$

c) $y(t) = \ln(\frac{1}{2}t^2 + e)$

Projects

2.4.1

a) If $s(t) = x/(x + e^{-t}(1 - x))$ then

$$s'(t) = \frac{xe^{-t}(1 - x)}{(x + e^{-t}(1 - x))^2} = s(1 - s),$$

and

$$s(0) = \frac{x}{x + 1 - x} = x.$$

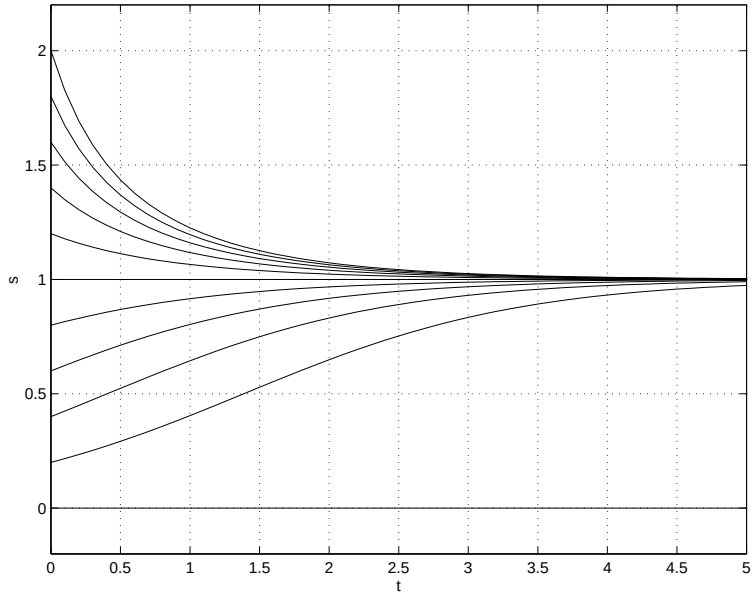

Figure 1: Solutions of $s(t)$ for different values of x. Differences in x seems to have little effect on the solution at high values of t, except for when $x = 0$.

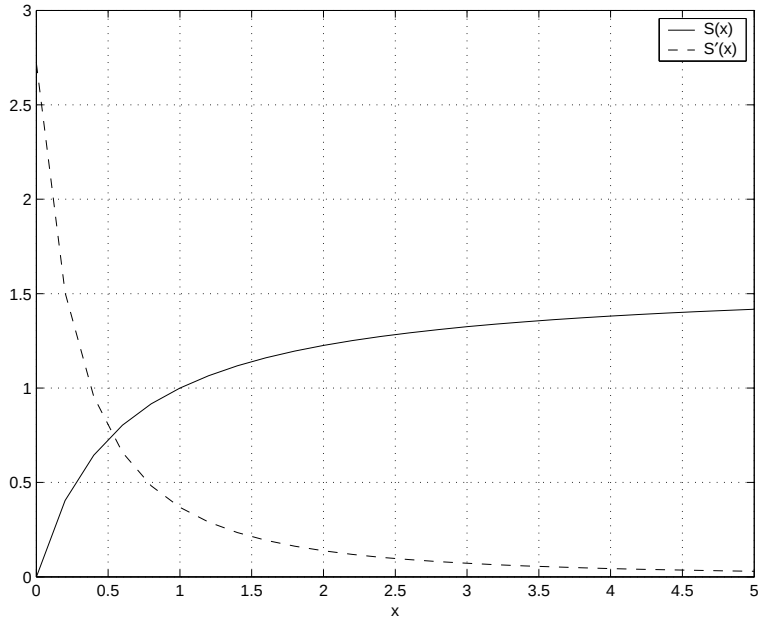

Figure 2: $S(x) = \frac{x}{x+e^{-1}(1-x)}$

b) See figure 1.

c) +d) See figure 2.

$$S'(x) = \frac{x + e^{-1}(1-x) - (1-e^{-1})x}{(x + e^{-1}(1-x))^2} = \frac{1}{e(e^{-1}x - x - e^{-1})^2}$$

e) We have from the Taylor series that $S(x+\varepsilon) \approx S(x) + \varepsilon S'(x)$. This means that small perturbations, ε, of the initial data will be more significant the greater $S'(x)$ is compared to $S(x)$. We see from figure 2 that $S'(x)$ is small compared to $S(x)$ exept at very low values of x. We conclude that small perturbations of the initial data will only have a significant effect on the solution close to $(x = 0)$. This is consistent with the results in figure 1.

f) If $u(t) = x/(x + e^t(1-x))$ then

$$u'(t) = \frac{-x(e^t(1-x)}{(x + e^t(1-x))^2} = u(u-1),$$

and

$$u(0) = \frac{x}{x + 1 - x} = x.$$

g) For $(0 < x < 1) : \lim_{t\to\infty} u(t)$ is x divided by infinity.
For $(x = 1) : u(t) = \frac{1}{1+0} = 1$.
For $(x > 1) : u\left(t \to \ln \frac{x}{x-1}\right)$ is x divided by $x - x$.

h) See figure 3.

i) $U(1) = 1, U(1.0000454) = 0.5000110$. Close to ($x$=1) the expression is extremely unstable with respect to small perturbations in x.

2.4.2

a) The Taylor series can be written as

$$u(t+k) = \sum_{n=0}^{3} \frac{u^{(n)}(t)}{n!}k^n + \frac{u^{(4)}(\xi)}{4!}k^4.$$

The final term is the *remainder term*, and is often simlpy written as $\mathcal{O}(k^4)$.

b)

$$\begin{aligned} u(t_{n+1}) &= u\left(\left((n + \frac{1}{2} + \frac{1}{2}\right)\Delta t\right) \\ &= u\left(\left(n + \frac{1}{2}\right)\Delta t\right) + \frac{\Delta t}{2}u'\left(\left(n + \frac{1}{2}\right)\Delta t\right) \\ &\quad + \frac{1}{2}\left(\frac{\Delta t}{2}\right)^2 u''\left(\left(n + \frac{1}{2}\right)\Delta t\right) + \frac{1}{6}\left(\frac{\Delta t}{2}\right)^3 u'''\left(\left(n + \frac{1}{2}\right)\Delta t\right) + \mathcal{O}(\Delta t^4) \end{aligned}$$

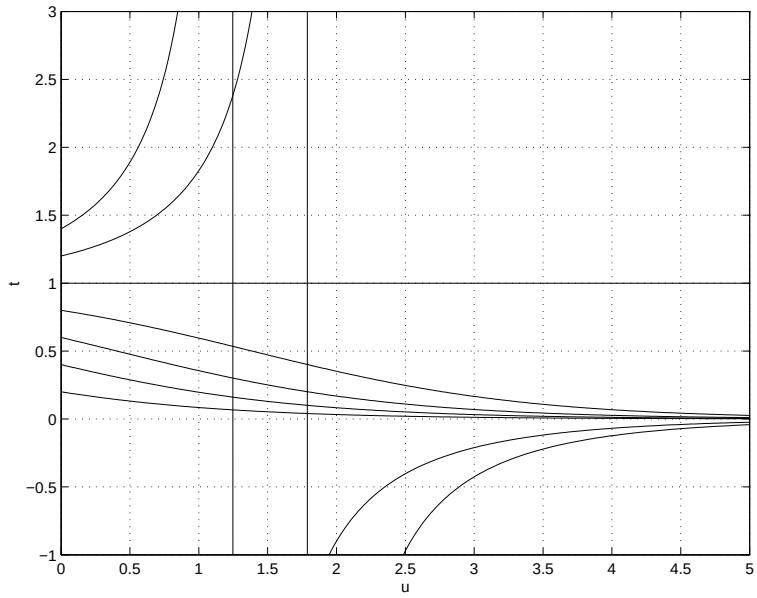

Figure 3: Solutions of $u(t)$ for different values of t. Differences in x seems to have little effect on the solution at high values of t, except for when $x = 1$.

$$\begin{aligned} u(t_n) &= u\left(\left(n+\frac{1}{2}-\frac{1}{2}\right)\Delta t\right) \\ &= u\left(\left(n+\frac{1}{2}\right)\Delta t\right) - \frac{\Delta t}{2}u'\left(\left(n+\frac{1}{2}\right)\Delta t\right) \\ &\quad +\frac{1}{2}\left(\frac{\Delta t}{2}\right)^2 u''\left(\left(n+\frac{1}{2}\right)\Delta t\right) - \frac{1}{6}\left(\frac{\Delta t}{2}\right)^3 u'''\left(\left(n+\frac{1}{2}\right)\Delta t\right) + \mathcal{O}(\Delta t^4) \end{aligned}$$

c) Combining the two equations from exercise **b** gives

$$u(t_{n+1}) - u(t_n) = \Delta t u'(t_{n+\frac{1}{2}}) + \frac{2}{6}\left(\frac{\Delta t}{2}\right)^3 u'''\left(\left(n+\frac{1}{2}\right)\Delta t\right) + \mathcal{O}_{n+1}(\Delta t^4) - \mathcal{O}_n(\Delta t^4).$$

We consider all but the first term to be the remainder, $\mathcal{O}(\Delta t^3)$, and divide by Δt on both sides of the equation, leading to

$$\frac{u(t_{n+1}) - u(t_n)}{\Delta t} = u'(t_{n+\frac{1}{2}}) + \mathcal{O}(\Delta t^2).$$

d) Equation (2.69) means we can replace $u'(t_{n+\frac{1}{2}})$ with $f(u(t_{n+\frac{1}{2}}))$ in the equation from exercise **c**.

e) We take the Taylor series of the functions $f(u(t_n))$ and $f(u(t_{n+1}))$:

$$f(u(t_n)) = f(u(t_{n+1/2})) + (-\frac{\Delta t}{2}u'(n+t_{1/2}) + \mathcal{O}(\Delta t^2))f'(u(t_{n+1/2})) + \mathcal{O}(\Delta t^2)$$

$$f(u(t_{n+1})) = f(u(t_{n+1/2})) + (\frac{\Delta t}{2} u'(n+t_{1/2}) + \mathcal{O}(\Delta t^2)) f'(u(t_{n+1/2})) + \mathcal{O}(\Delta t^2)$$

Combining the two equations gives

$$\frac{1}{2}(f(u(t_n)) + f(u(t_{n+1}))) = f(u(t_{n+1/2})) + \mathcal{O}(\Delta t^2).$$

Combining this with the result from exercise **c** we find that

$$\begin{aligned} \frac{u(t_{n+1}) - u(t_n)}{\Delta t} &= u'(t_{n+\frac{1}{2}}) + \mathcal{O}(\Delta t^2) \\ &= f(u(t_{n+\frac{1}{2}})) + \mathcal{O}(\Delta t^2) \\ &= \frac{1}{2}(f(u(t_n)) + f(u(t_{n+1}))) + \mathcal{O}(\Delta t^2). \end{aligned}$$

f) Removing $\mathcal{O}(\Delta t^2)$ from the equation in **e** gives us the approximation $u_n \approx u(t_n)$:

$$\begin{aligned} \frac{u_{n+1} - u_n}{\Delta t} &= \frac{f(u_{n+1}) + f(u_n)}{2} \\ u_{n+1} - \frac{\Delta t}{2} f(u_{n+1}) &= u_n + \frac{\Delta t}{2} f(u_n) \end{aligned}$$

g) See the program **pro242.java**. At time $T = 5$ the results are:

N	(error explicit Euler)/Δt	(error imlicit Euler)/Δt	(error Crank-Nicolson)/Δt^2
50	2.09	3.07	0.42
100	2.27	2.76	0.42
200	2.38	2.62	0.42

h) See the program **pro242h.java**. At time $T = 1$ the results are:

N	(error explicit Euler)/Δt	(error imlicit Euler)/Δt	(error Crank-Nicolson)/Δt^2
20	1.89	1.75	3.31
40	1.80	1.76	3.23
80	1.78	1.77	3.21

i) In general,

$$\frac{u(t_{n+1}) - u(t_n)}{\Delta t} = \frac{1}{2}(f(u(t_n)) + f(u(t_{n+1}))) + \mathcal{O}(\Delta t^2).$$

We can approximate $f(u(t_{n+1}))$ by $f(u_n + \Delta t f(u_n))$. The error in this approximation is $\mathcal{O}(\Delta t^2)$. This produces the formula

$$u_{n+1} = u_n + \frac{\Delta t}{2}\left[f(u_n)) + f(u_n + \Delta t f(u_n))\right].$$

This is known as the *Heun scheme*.

j) See the program **pro242j.java**.

Repeating the experiments in exercise **g** ($T = 5$):

N	(error Heun)/Δt^2
50	0.77
100	0.80
200	0.82

Repeating the experiments in exercise **h** ($T = 1$):

N	(error Heun)/Δt^2
20	2.59
40	2.45
80	2.33

k) The explicit Euler scheme can be derived from the Taylor series of $u(t_n + \Delta t)$:

$$u(t_{n+1}) = u(t_n + \Delta t) = u(t_n) + \Delta t u'(t_n) + \mathcal{O}(\Delta t^2)$$

This leads to the approximation

$$\begin{aligned} u_{n+1} &= u_n + \Delta t u'_n \\ &= u_n + f(t_n, u_n). \end{aligned}$$

The implicit Euler scheme is derived in a similar fashion:

$$u(t_n) = u(t_{n+1} - \Delta t) = u(t_{n+1}) - \Delta t u'(t_{n+1}) + \mathcal{O}(\Delta t^2)$$

This leads to the approximation

$$\begin{aligned} u_n &= u_{n+1} - \Delta t u'_{n+1} \\ &= u_{n+1} - \Delta t f(t_{n+1}, u_{n+1}). \end{aligned}$$

The Crank-Nicholson scheme is derived as in exercise **f**, replacing $f(u_n)$ with $f(t_n, u_n)$ and $f(u_{n+1})$ with $f(t_{n+1}, u_{n+1})$.

The Heun scheme is derived as in exercise **i**, replacing $f(u_n)$ with $F_1 = f(t_n, u_n)$, and $f(u_n + \delta t f(u_n))$ with $F_2 = f(t_{n+1}, u_n + \Delta t F_1$.

l) We adjust the program **pro242j.java** so that it uses the fourth order Runge-Kutta scheme, giving the following results:

Repeating the experiments in exercise **g** ($T = 5$):

N	error/Δt	error/Δt^2	error/Δt^3	error/Δt^4	error/Δt^5
50	$3.83 \cdot 10^{-5}$	$3.83 \cdot 10^{-4}$	$3.83 \cdot 10^{-3}$	$3.83 \cdot 10^{-2}$	0.383
100	$5.00 \cdot 10^{-6}$	$9.91 \cdot 10^{-4}$	$2.00 \cdot 10^{-3}$	$4.00 \cdot 10^{-2}$	0.80
200	$6.38 \cdot 10^{-7}$	$2.55 \cdot 10^{-5}$	$1.02 \cdot 10^{-3}$	$4.08 \cdot 10^{-2}$	1.63

Repeating the experiments in exercise **h** ($T = 1$):

N	error/Δt	error/Δt^2	error/Δt^3	error/Δt^4	error/Δt^5
40	$8.33 \cdot 10^{-5}$	$3.33 \cdot 10^{-3}$	0.133	5.33	213
80	$1.15 \cdot 10^{-5}$	$9.19 \cdot 10^{-4}$	$7.36 \cdot 10^{-2}$	5.88	471
160	$1.46 \cdot 10^{-6}$	$2.34 \cdot 10^{-4}$	$3.75 \cdot 10^{-2}$	6.00	959

m) If $u(t) = e^{-(\frac{1-t}{\varepsilon})^2}$ then

$$u'(t) = e^{-(\frac{1-t}{\varepsilon})^2}(\frac{2}{\varepsilon^2} - \frac{2t}{\varepsilon^2}) = u\frac{2(1-t)}{\varepsilon^2},$$

and

$$u(0) = e^{-(\frac{1-0}{\varepsilon})^2} = e^{\frac{-1}{\varepsilon^2}}.$$

n) The program **pro242n.java** can be used to compare the schemes. (**NOTE**: If one uses a different funcion f, the Crank-Nicolson and Implicit Euler segments of the program will probably need adjustment.)

Programs (MATLAB and JAVA)

ex21.m

```
function ex21(N,T)
% Solves the expression from exercise 2.1,
% using both explicit and implicit schemes.
% Compares with exact solution.
%
%  example: ex21(50,1)

Dt=T/N;
t=0:Dt:T;

y(1)=0; z(1)=0; r(1)=0;

for i= 2:(N+1);
     t(i)=t(i-1)+Dt;
     y(i)= y(i-1)+  Dt*(1+4*y(i-1));
     z(i)=(z(i-1) +Dt)/(1-4*Dt);
     r(i)= (exp(4*t(i))-1)/4;
end

plot(t,r, t,y,'r--',t,z,'g-.');
legend('exact solution','explicit scheme','implicit scheme',2);

% Errors:

Ee= abs(y(N+1)-r(N+1));
disp(sprintf('Error explicit/Delta t: %g ',Ee/Dt));

Ei=abs(z(N+1)-r(N+1));
disp(sprintf('Error implicit/Delta t: %g ',Ei/Dt));
```

ex22.m

```
function ex22(N,T)
% Solves the expression from exercise 2.2,
```

```
% using both explicit and impicit schemes.
% Compares with exect solution.
%
% example: ex22(50,1)


Dt=T/N;
t=0:Dt:T;

y(1)=1;
z(1)=1;
r(1)=1;

for i= 2:(N+1);
     t(i)=t(i-1)+Dt;
     y(i)= y(i-1)+  Dt*(y(i-1)+2*t(i-1)-t(i-1)*t(i-1));
     z(i)=(z(i-1)+Dt*(2*t(i)-t(i)*t(i)))/(1-Dt);
r(i)= exp(t(i))+t(i)*t(i);
end

plot(t,r, t,y,'r--',t,z,'g-.');
legend('exact solution','explicit scheme','implicit scheme',2);

% Errors:

Ee= abs(y(N+1)-r(N+1));
disp(sprintf('Error explicit/Delta t: %g ',Ee/Dt));

Ei=abs(z(N+1)-r(N+1));
disp(sprintf('Error implicit/Delta t: %g ',Ei/Dt));
```

ex25.m

```
function f  = ex25(R)
% Calculates the population of the United States,
% using the both the exponential model
% and the logistic model.
%
% example: ex25(290)

r0=5.3; a=0.03;

% exponential model:
     t=1800:1900;
     r=r0 * exp(a*(t-1800));

% actual population
 Y=1800:10:1900;
P=[5.3, 7.2, 9.6, 12.9, 17, 23.2, 31.4, 38.6, 50.2, 63, 76.2];
```

```
% logistic model:
l= r0*R./(r0 + (R - r0)*exp(-a*(t-1800)));

plot(t,r,t,l,'g--',Y,P,'rx');
  xlabel('year'); ylabel('population (millions)');
title('Population of the United States');
legend('exponential growth model',
       ['logistic model, R=',int2str(R)],'actual population',2);
```

ex27.java

```
/* Program for exercise 2.7.
 * uses three different schemes to solve (2.38)
 * and compares the solutions to the exact solution.
 * example: java ex27 1 100
 */

class ex27
{
    public static void main(String[] args)
    {
        double N, T;
        double u, u1, u2, u3, Dt, Et, Em, Es;

        T = Double.parseDouble(args[0]);
        N = Double.parseDouble(args[1]);

        System.out.print("T= " +T+" N= "+N+"\n");
        Dt=T/N;
        System.out.print("Dt= "+ Dt+"\n");

        u=-10/(1+10*T);

        u1=-10;
        for (int i=1; i<N+1; i++) {
            u1=( 1 - Math.sqrt(1- Dt*(Dt*u1*u1 +2*u1)))/Dt;

        }
        System.out.print("Trapes: u= "+ u1 +"\n");

        u2=-10;
        for (int i=1; i<N+1; i++) {
            u2=( 2 - Dt*u2 - 2*Math.sqrt(1- 2*Dt*u2))/Dt;
           }
        System.out.print("Midpoint: u= "+ u2 +"\n");

        u3=-10;
        for (int i=1; i<N+1; i++) {
         u3=3*( 1-((Dt*u3)/3)- Math.sqrt(1- Dt*Dt*u3*u3/3 -2*Dt*u3))/(2*Dt);
```

```
            }
        System.out.print("Simpson: u= "+ u3 +"\n");

        Et=(u1 - u);
        Em=(u2 - u);
        Es=(u3 - u);
        System.out.print("Exact solution: "+ u +"\n");
        System.out.print("Error/Dt*Dt (Trapes): "+ Et/(Dt*Dt) +"\n");
        System.out.print("Error/Dt*Dt (Midpoint): "+ Em/(Dt*Dt) +"\n");
        System.out.print("Error/Dt*Dt (Simpson): "+ Es/(Dt*Dt) +"\n");
        }
     }
```

pro242.java

```
/*  Program for Project 2.4.2.
 *  exercise g): u'=u, u(0)=1.
 *  Solves u at time T in three ways:
 *  explicit Euler (u1)
 *  implicit Euler (u2)
 *  and Crank-Nicolson (u3)
 *  example: java pro242 5 100
 */


class pro242
{
    public static void main(String[] args)
    {
        double N, T;
        double u1, u2, u3, Dt, k, E_exp, E_imp, E_CN;

        T = Double.parseDouble(args[0]);
        N = Double.parseDouble(args[1]);

        System.out.print("T: " +T+" N: "+N+"\n");
        Dt=T/N;
        System.out.print("Dt: "+ Dt+"\n");

        u1=1;
        u2=1;
        u3= 1;
        for (int i=1; i<N+1; i++) {
            u1= u1 + u1*Dt;
            u2= u2/(1-Dt);
            u3= u3*(1 + Dt*0.5)/(1 - Dt*0.5);

        }
        System.out.print("Explicit Euler: u= "+ u1 +"\n");
        System.out.print("Implicit Euler: u= "+ u2 +"\n");
```

```
        System.out.print("Crank-Nicholson "+ u3 +"\n");

        k=Math.exp(T);

        E_exp=Math.abs(u1-k)/k;
        E_imp=Math.abs(u2-k)/k;
        E_CN=Math.abs(u3-k)/k;

        System.out.print("Exact solution: "+ k +"\n");
        System.out.print("Error explicit/Dt  "+ E_exp/Dt +"\n");
        System.out.print("Error implicit/Dt "+E_imp/Dt  +"\n");
        System.out.print("Error Crank/Dt  "+E_CN/Dt +"\n");
        System.out.print("Error Crank/Dtˆ2  "+E_CN/(Dt*Dt) +"\n");
        }
    }
```

pro242h.java

```
/*  Program for Project 2.4.2.
 *  exercise h): u'=u(1-u), u(0)=10.
 *  Solves u at time T in three ways:
 *  explicit Euler (u1)
 *  implicit Euler (u2)
 *  and Crank-Nicolson (u3)
 *  example: java pro242h 1 20
 */

class pro242h
{
 public static void main(String[] args)
 {
 double N, T;
 double u1, u2, u3, Dt, k, C, E_exp, E_imp, E_CN;

 T = Double.parseDouble(args[0]);
 N = Double.parseDouble(args[1]);

 System.out.print("T: " +T+" N: "+N+"\n");
 Dt=T/N;
 C=10.0/9.0;
 System.out.print("Dt: "+ Dt+"\n");

 u1=10;
 u2=10;
 u3= 10;
 for (int i=1; i<N+1; i++) {
   u1= u1 + u1*Dt*(1-u1);
   u2=(Dt -1 + Math.sqrt(1 - 2*Dt +4*Dt*u2 +Dt*Dt))/(2*Dt);
   u3=(0.5*Dt-1+Math.sqrt(1-Dt+Dt*Dt*0.25+2*Dt*u3+Dt*Dt*u3*(1-u3)))/Dt;
     }
```

```
 System.out.print("Explicit Euler: u= "+ u1 +"\n");
 System.out.print("Implicit Euler: u= "+ u2 +"\n");
 System.out.print("Crank-Nicholson "+ u3 +"\n");

 k=(C*Math.exp(T))/(C*Math.exp(T)-1);

 E_exp=Math.abs(u1-k);
 E_imp=Math.abs(u2-k);
 E_CN=Math.abs(u3-k);

 System.out.print("Exact solution: "+ k +"\n");
 System.out.print("Error explicit/Dt  "+ E_exp/Dt +"\n");
 System.out.print("Error implicit/Dt "+E_imp/Dt  +"\n");
 System.out.print("Error Crank/Dt  "+E_CN/Dt +"\n");
 System.out.print("Error Crank/Dt^2  "+E_CN/(Dt*Dt) +"\n");
     }
  }
```

pro242n.java

```
/* Program for Project 2.4.2.
 * exercise n)
 * Solves u at time 'Time' in five different ways.
 * measures how long time it takes, and finds the error
 * example: java pro242 1.1 20000
 */
class pro242n
{
  public static void main(String[] args)
  {
   double N, Time;
   double startTime, endTime;
   double u, u0, Dt, solution, F1, F2, F3, F4;
   double epsilon=0.25;

   Time = Double.parseDouble(args[0]);
   N = Double.parseDouble(args[1]);

   Dt=Time/N;
   u0 = Math.exp(-1/(epsilon*epsilon));
   solution = Math.exp(-(1-Time)*(1-Time)/(epsilon*epsilon));

   System.out.print("Exact solution: "+ solution +"\n");

// Explicit Euler
   startTime = System.currentTimeMillis();
   u=u0;
   for (int i=1; i<N+1; i++) {
     u=u+Dt*f(i*Dt,u,epsilon);
        }
```

```
   endTime = System.currentTimeMillis();
   System.out.print("Explicit Euler: T= "+ Time + " N= "+ N+ "\n");
   System.out.print("  solution: "+ u +"\n");
   System.out.print("  absolute error: "+ Math.abs(u-solution) +"\n");
   System.out.print("  time (milliseconds): "+ (endTime -startTime) +"\n");

// Implicit Euler
   startTime = System.currentTimeMillis();
   u=u0;
   for (int i=1; i<N+1; i++) {
     u=u/(1-Dt*f((i+1)*Dt,1,epsilon));
        }
   endTime = System.currentTimeMillis();
   System.out.print("Implicit Euler: T= "+ Time + " N= "+ N+ "\n");
   System.out.print("  solution: "+ u +"\n");
   System.out.print("  absolute error: "+ Math.abs(u-solution) +"\n");
   System.out.print("  time (milliseconds): "+ (endTime -startTime) +"\n");

// Crank-Nicolson
   startTime = System.currentTimeMillis();
   u=u0;
   for (int i=1; i<N+1; i++) {
     u=(u+Dt*f(Dt*i,u,epsilon)/2)/(1-Dt*f((i+1)*Dt,1,epsilon)/2);
        }
    endTime = System.currentTimeMillis();
    System.out.print("Crank-Nicolson: T= "+ Time + " N= "+ N+ "\n");
    System.out.print("  solution: "+ u +"\n");
    System.out.print("  absolute error: "+ Math.abs(u-solution) +"\n");
    System.out.print("  time (milliseconds): "+ (endTime -startTime) +"\n");

// Heun
   startTime = System.currentTimeMillis();
   u=u0;
   for (int i=1; i<N+1; i++) {
     F1=f(i*Dt,u,epsilon);
     F2=f((i+1)*Dt,u+Dt*F1,epsilon);
     u=u+Dt*(F1+F2)/2;
        }
   endTime = System.currentTimeMillis();
   System.out.print("Heun: T= "+ Time + " N= "+ N+ "\n");
   System.out.print("  solution: "+ u +"\n");
   System.out.print("  absolute error: "+ Math.abs(u-solution) +"\n");
   System.out.print("  time (milliseconds): "+ (endTime -startTime) +"\n");

// Runge-Kutta
   startTime = System.currentTimeMillis();
   u=u0;
   for (int i=1; i<N+1; i++) {
     F1=f(i*Dt,u,epsilon);
     F2=f((i+0.5)*Dt,u+Dt*F1/2,epsilon);
```

```
     F3=f((i+0.5)*Dt,u+Dt*F2/2,epsilon);
     F4=f((i+1)*Dt,u+Dt*F3,epsilon);
     u=u+Dt*(F1+2*F2+2*F3+F4)/6;
        }
   endTime = System.currentTimeMillis();
   System.out.print("Runge-Kutta: T= "+ Time + " N= "+ N+ "\n");
   System.out.print("  solution: "+ u +"\n");
   System.out.print("  absolute error: "+ Math.abs(u-solution) +"\n");
   System.out.print("  time (milliseconds): "+ (endTime -startTime) +"\n");
   }

public static double f(double t, double u, double eps)
{
    double res=u*2*(1-t)/(eps*eps);
    return res;
}
}
```