Math 2250−4/2280−1
Numerical Methods, sections 2.4−2.6
SOLUTIONS to numerical questions

2.4.29. (This problem involves work done by hand in part 29a.) This problem illustrates how a numerical algorithm may accidently pass right through an x−value where the real solution blows up.
29a) I'll do the hand work first. This is a separable DE, and I solve it below:

$$7\cdot x\cdot\left(\frac{dy}{dx}\right)=-y$$

$$\frac{dy}{y}=-\frac{1}{7}x\cdot dx$$

$$\ln(|y|)=-\frac{1}{7}\cdot\ln\left(|x|\right)+C_1$$

$$y=C\cdot x^{\left(-\frac{1}{7}\right)}$$

Since y(−1)=−1, we deduce

$$y=-\frac{1}{x^{\frac{1}{7}}}.$$

Notice that the actual continuous solution to this IVP blows up (approaches infinity) as x approaches zero from the left, so does not extend as a solution to the IVP to non−negative x! As we shall see below, uncareful use of Euler fails to detect this fact and merrily continues approximating right through the singularity!
29b) Euler, with h=0.15, from −1 to .5 (so n=10)

```
> restart:
  Digits:=6: #that should be enough accuracy
  x0:=-1.0:xn:=0.5:
  y0:=1:
  f:=(x,y)->-y/(7*x);#slope function
  g:=(x,y)->abs(x)^(-1/7.);
     #you didn't need to this, but I'll compare
     #to actual solution, and if I don't write
     #it in a strange way Maple tries to take
     #complex roots
```

$$f:=(x,y)\rightarrow-\frac{1}{7}\,\frac{y}{x}$$

$$g:=(x,y)\rightarrow|x|^{-\frac{1}{7.}}$$ **(1)**

```
> n:=10;h:=(xn-x0)/n;
```

$$n:=10$$

$$h:=0.150000$$ **(2)**

```
> x:=x0:y:=y0:
  for i from 1 to n do
            k:= f(x,y):  #current slope,use: to suppress output
            y:= y + h*k: #new y value via Euler
            x:= x + h:   #updated x-value:
            print(x,y,g(x));
```

```
        #display current values,
        #and compare to exact solution.
od:
```

−0.850000, 1.02143, 1.02349

−0.700000, 1.04718, 1.05227

−0.550000, 1.07924, 1.08916

−0.400000, 1.12129, 1.13985

−0.250000, 1.18136, 1.21901

−0.100000, 1.28262, 1.38950

0.050000, 1.55747, 1.53413

0.200000, 0.889984, 1.25850

0.350000, 0.794629, 1.16180

0.500000, 0.745978, 1.10409 **(3)**

Notice that there is a huge jump in the "real" solution between x=−0.1 and 0.05, reflecting the actual discontinuity. (And, in fact, there is no continuous solution which bridges x=0, so we shouldn't even talk about a "real" solution after that vertical asymptote at x=0. But Euler looks fine.

29c) h=0.03:

```
> n:=50;h:=(xn-x0)/n;
  x:=x0:y:=y0:
  for i from 1 to n do
          k:= f(x,y):  #current slope,use: to suppress output
          y:= y + h*k: #new y value via Euler
          x:= x + h:   #updated x-value:
            if frac(i/5)=0 then print(x,y,g(x));
            fi:
      od:
```

$n := 50$

$h := 0.0300000$

−0.850000, 1.02306, 1.02349

−0.700000, 1.05120, 1.05227

−0.550000, 1.08704, 1.08916

−0.400000, 1.13579, 1.13985

−0.250000, 1.21041, 1.21901

−0.100000, 1.36128, 1.38950

0.0500000, 1.87220, 1.53413

0.200000, 1.47120, 1.25850

0.350000, 1.35068, 1.16180

0.500000, 1.28080, 1.10409 **(4)**

Now there's mabye just a tiny hint from Euler that something strange is happening near x=0, since there's a relatively large jump in the Euler approximation between x=−.1 and x=−.05.

29c) continuing, take h=0.006:

```
> n:=250;h:=(xn-x0)/n;
```

$$n := 250$$

$$h := 0.00600000 \quad \textbf{(5)}$$

```
> x:=x0:y:=y0:
  for i from 1 to n do
            k:= f(x,y):  #current slope,use: to suppress output
            y:= y + h*k: #new y value via Euler
            x:= x + h:   #updated x-value:
            if frac(i/25)=0 then print(x,y,g(x));
              fi:
      od:
```

$$-0.850000, 1.02340, 1.02349$$
$$-0.700000, 1.05204, 1.05227$$
$$-0.550000, 1.08872, 1.08916$$
$$-0.400000, 1.13901, 1.13985$$
$$-0.250000, 1.21724, 1.21901$$
$$-0.100000, 1.38346, 1.38950$$
$$0.0500000, 1.00846, 1.53413$$
$$0.200000, 0.821023, 1.25850$$
$$0.350000, 0.757141, 1.16180$$
$$0.500000, 0.719223, 1.10409 \quad \textbf{(6)}$$

Looked at in isolation this latest Euler approximation wouldn't indicate anything is wrong. But if you compare it to the h= 0.03 approximation we see that as x increases from x=−1 to x=−0.1 both approximations are very close, but that there is a large difference in the x=0.05 approximate values, which are 1.87 and 1.01, respectively. This would be a strong hint to look more closely at what's going on near x=0!!!

problem #5 in sections 2.4, 2.5, 2.6 homework. This is the same initial value problem in each case, studied successively with Euler, improved Euler, and Runge−Kutta.

```
> restart:
  x0:=0.0:xn:=0.5:
  y0:=1:
  f:=(x,y)->y-x-1;#slope function
  g:=x->2+x-exp(x);#solution function
```

$$f := (x, y) \rightarrow y - x - 1$$

$$g := x \rightarrow 2 + x - e^{x} \quad \textbf{(7)}$$

Euler with n=2: (2.4#5)

```
> Digits:=5: #to get three or four decimal places accurately
> n:=2:h:=(xn-x0)/n:
> x:=x0:y:=y0:
  for i from 1 to n do
            k:= f(x,y):  #current slope,use: to suppress output
            y:= y + h*k: #new y value via Euler
            x:= x + h:   #updated x-value:
```

```
            print(x,y,g(x));
              #display current values,
              #and compare to exact solution.
        end do:
```

0.25000, 1., 0.9660

0.50000, 0.93750, 0.8513 **(8)**

Euler with n=5: (2.4#5)

```
> n:=5:h:=(xn-x0)/n:x:=x0:y:=y0:
  for i from 1 to n do
            k:= f(x,y):  #current slope,use: to suppress output
            y:= y + h*k: #new y value via Euler
            x:= x + h:   #updated x-value:
            print(x,y,g(x));
              #display current values,
              #and compare to exact solution.
        end do:
```

0.10000, 1., 0.9948

0.20000, 0.99000, 0.9786

0.30000, 0.96900, 0.9501

0.40000, 0.93590, 0.9082

0.50000, 0.88949, 0.8513 **(9)**

```
> .8513-.9375; #n=2 error
  .8513-.8895; #n=5 error
```

−0.0862

−0.0382 **(10)**

2.5#5: improved Euler with h=0.1

```
> Digits:=5:
  n:=5:h:=(xn-x0)/n:x:=x0:y:=y0:
  for i from 1 to n do
      k1:=f(x,y):          #left-hand slope
      k2:=f(x+h,y+h*k1):   #approximation to right-hand slope
      k:= (k1+k2)/2:       #approximation to average slope
      y:= y+h*k:            #improved Euler update
      x:= x+h:             #update x
      print(x,y,g(x));
  od:
```

0.10000, 0.99500, 0.9948

0.20000, 0.97898, 0.9786

0.30000, 0.95077, 0.9501

0.40000, 0.90910, 0.9082

0.50000, 0.85256, 0.8513 **(11)**

```
> .8513-.85256; #error (notice it beats Euler with n=5)
```

−0.00126 **(12)**

2.6#5: Runge Kutta with h=0.25

```
> Digits:=6: #to get 5-6 decimal places
  n:=2:h:=(xn-x0)/n:

> x:=x0:y:=y0:
> for i from 1 to n do
      k1:=f(x,y):              #left-hand slope
      k2:=f(x+h/2,y+h*k1/2):   #1st guess at midpoint slope
      k3:=f(x+h/2,y+h*k2/2):   #second guess at midpoint slope
      k4:=f(x+h,y+h*k3):       #guess at right-hand slope
      k:=(k1+2*k2+2*k3+k4)/6:  #Simpson's approximation for the integral
      x:=x+h:                  #x update
      y:=y+h*k:                #y update
      print(x,y,g(x));              #display current values
   end do:
```

0.250000, 0.965983, 0.96597

0.500000, 0.851301, 0.85128 **(13)**

```
> .85128-.851300; #error -notice it beats IE with n=5
```

-0.000020 **(14)**

2.6 famous numbers page 145 via Runge Kutta. We'll do the doubling of steps by nesting two do loops:
a) *e*:

```
> restart :
> Digits := 20 : #so we can see when we get nine decimal places of accuracy
> x0 := 0. :y0 := 1. : xn := 1. :
  f := (x, y) →y : # slope function
> n := 10 : #initial number of subdivisions
> for j from 1 to 6 do #use j to double subdivisions in succession
  x := x0 :y := y0 :h := (xn − x0)/n :
    for i from 1 to n do
     k1 := f(x, y) :          #left−hand slope
     k2 := f(x + h/2, y + h * k1/2) : #1st guess at midpoint slope
     k3 := f(x + h/2, y + h * k2/2) : #second guess at midpoint slope
     k4 := f(x + h, y + h * k3) :    #guess at right−hand slope
     k := (k1 + 2 * k2 + 2 * k3 + k4)/6 : #Simpson's approximation for the integral
     x := x + h :             #x update
     y := y + h * k :         #y update
    end do:                 # end of Runge Kutta loop
   print(x, y, exp(1.)); #print current approximation
  n := 2·n : #double the number of steps and repeat as j goes from 1 to 6
  end do:
```

1.00000000000000000000, 2.71827974413516565390, 2.71828182845904523540

1.00000000000000000000, 2.71828169265633395700, 2.71828182845904523540

1.00000000000000000000, 2.71828181979285606720, 2.71828182845904523540

1.00000000000000000000, 2.71828182791173942330, 2.71828182845904523540

1.00000000000000000000, 2.71828182842466003630, 2.71828182845904523540

1.00000000000000000000, 2.71828182845685695710, 2.71828182845904523540 **(15)**

>

To me, 9 decimal places would mean the 10^{-9}position, which is thru 2.718281828 . If we allow round off we get this accuracy starting with n = 80 (the fourth row).

b) $\ln(2)$: I copy and paste, and make minor changes

```
> restart :
> Digits := 20 : #so we can see when we get nine decimal places of accuracy
> x0 := 1. :y0 := 0. : xn := 2. :
  f := (x, y) → 1/x : # slope function
> n := 10 : #initial number of subdivisions
> for j from 1 to 6 do #use j to double subdivisions in succession
  x := x0 :y := y0 :h := (xn − x0)/n :
    for i from 1 to n do
     k1 := f(x, y) :          #left−hand slope
     k2 := f(x + h/2, y + h * k1/2) : #1st guess at midpoint slope
     k3 := f(x + h/2, y + h * k2/2) : #second guess at midpoint slope
     k4 := f(x + h, y + h * k3) :    #guess at right−hand slope
     k := (k1 + 2 * k2 + 2 * k3 + k4)/6 : #Simpson's approximation for the integral
     x := x + h :             #x update
     y := y + h * k :          #y update
    end do:                 # end of Runge Kutta loop
   print(x, y, ln(2.)); #print current approximation
  n := 2·n : #double the number of steps and repeat as j goes from 1 to 6
  end do:
```

2.0000000000000000000, 0.69314737466511611898, 0.69314718055994530942

2.0000000000000000000, 0.69314719274795602655, 0.69314718055994530942

2.0000000000000000000, 0.69314718132258694690, 0.69314718055994530942

2.0000000000000000000, 0.69314718060762436944, 0.69314718055994530942

2.0000000000000000000, 0.69314718056292546894, 0.69314718055994530942

2.0000000000000000000, 0.69314718056013157279, 0.69314718055994530942 **(16)**

>

We want to get and error of less than $0.5\cdot10^{-9}$, within 0.6931471805 and this happens starting at n = 80. (4th row again).

c) π :

```
> restart :
> Digits := 20 : #so we can see when we get nine decimal places of accuracy
> x0 := 0. :y0 := 0. : xn := 1. :
  f := (x, y) → 4./(x^2 + 1) : # slope function
> n := 10 : #initial number of subdivisions
> for j from 1 to 6 do #use j to double subdivisions in succession
  x := x0 :y := y0 :h := (xn − x0)/n :
    for i from 1 to n do
     k1 := f(x, y) :          #left−hand slope
     k2 := f(x + h/2, y + h * k1/2) : #1st guess at midpoint slope
     k3 := f(x + h/2, y + h * k2/2) : #second guess at midpoint slope
     k4 := f(x + h, y + h * k3) :    #guess at right−hand slope
```

```
    k := (k1 + 2 * k2 + 2 * k3 + k4) / 6 : #Simpson's approximation for the integral
    x := x + h :              #x update
    y := y + h * k :          #y update
  end do:                  # end of Runge Kutta loop
 print(x, y, evalf(π)); #print current approximation
n := 2·n : #double the number of steps and repeat as j goes from 1 to 6
end do:
```

1.0000000000000000000, 3.1415926529697849896, 3.1415926535897932385

1.0000000000000000000, 3.1415926535801051491, 3.1415926535897932385

1.0000000000000000000, 3.1415926535896418615, 3.1415926535897932385

1.0000000000000000000, 3.1415926535897908734, 3.1415926535897932385

1.0000000000000000000, 3.1415926535897932019, 3.1415926535897932385

1.0000000000000000000, 3.1415926535897932385, 3.1415926535897932385 **(17)**

>

We want to get and error of less than $0.5 \cdot 10^{-9}$, within 3.1415926535 and this happens starting at n = 20. (second row).