```
> restart;
```

Stability properties of Euler-like methods

In this worksheet we are going to be considering the numeric solution of the following initial value problem (which is analytically solvable):

```
> IVP := [diff(y(x),x)=lambda*y(x),y(0)=1];
  dsolve(IVP);
```

$$IVP := \left[\frac{d}{dx} y(x) = \lambda\, y(x), y(0) = 1 \right]$$

$$y(x) = e^{\lambda x} \tag{1}$$

The three stencils we consider for the numeric solution are the forward and backward Euler schemes, and the trapezoidal method. For this problem, each stencil gives the value of y(x+h) = y_new explicitly in terns of y(x) = y_old.

```
> forward := (y_old, lambda, h) -> y_old*(1+lambda*h);
  backward := (y_old, lambda, h) -> -y_old/(-1+lambda*h);
  trap := (y_old, lambda, h) -> -y_old*(2+lambda*h)/(-2+lambda*h);
```

$$forward := (y\_old, \lambda, h) \rightarrow y\_old\,(1 + \lambda h)$$

$$backward := (y\_old, \lambda, h) \rightarrow -\frac{y\_old}{-1 + \lambda h}$$

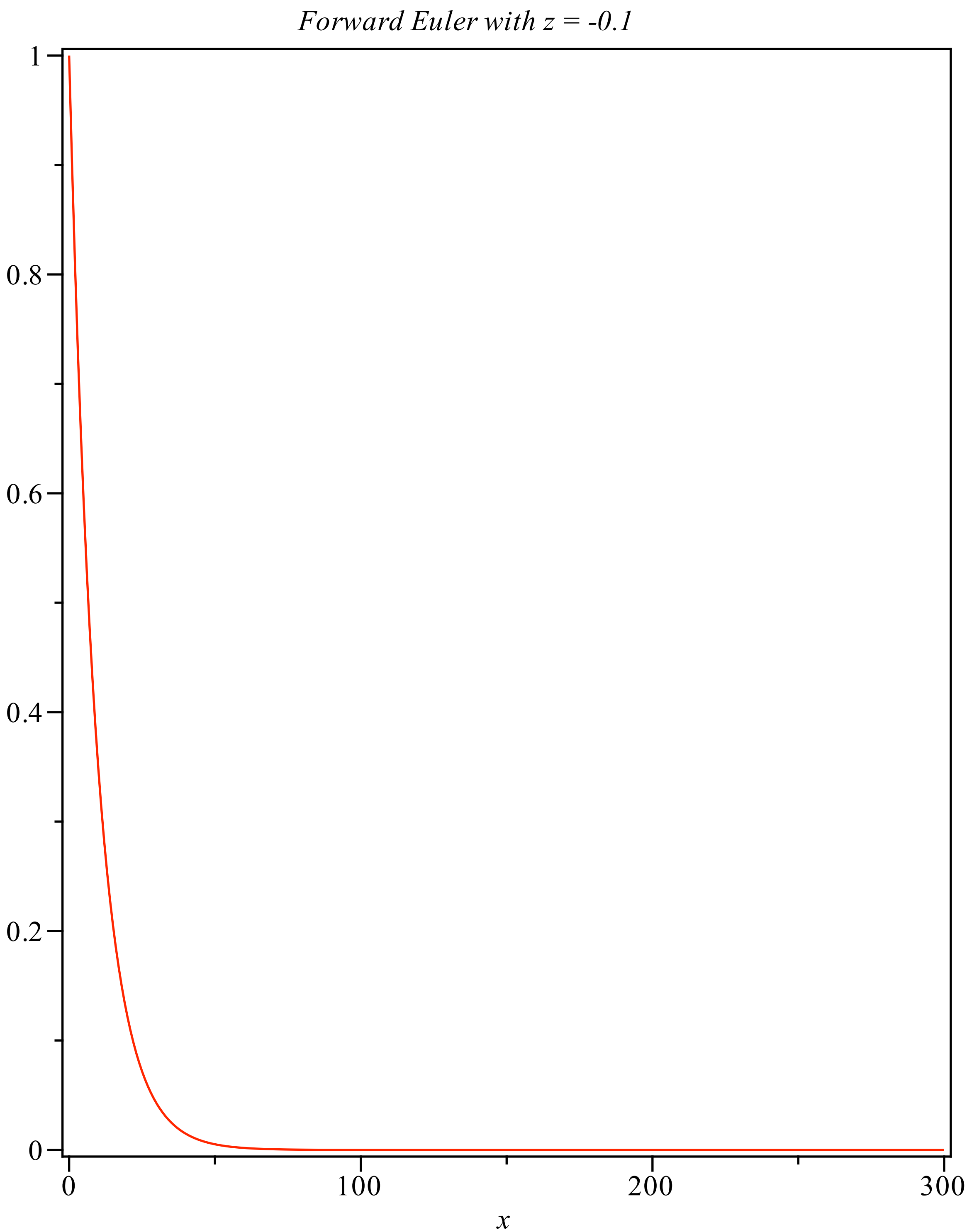
$$trap := (y\_old, \lambda, h) \rightarrow -\frac{y\_old\,(2 + \lambda h)}{-2 + \lambda h} \tag{2}$$

Since each stencil only involves lambda*h = z, the numerical solutions a parameterized by z only. The following procedure generates the numerical solution for 0 < x < xf=30/|lambda| for a given choice of stencil and z:

```
> EULER := proc(z,stencil)
         local x, y, i,h,lambda,xf,N:
         h := 1:
         lambda := z/h:
         xf := 30*abs(1/lambda):
         N := round(xf/h);
         x := Array(0..N,[seq(evalf(i*h),i=0..N)]):
         y := Array(0..N):
         y[0] := 1:
         for i from 1 to N do:
             y[i] := evalf(stencil(y[i-1],lambda,h)):
         od:
         [seq([x[i],y[i]],i=0..N)]:
  end proc:
```

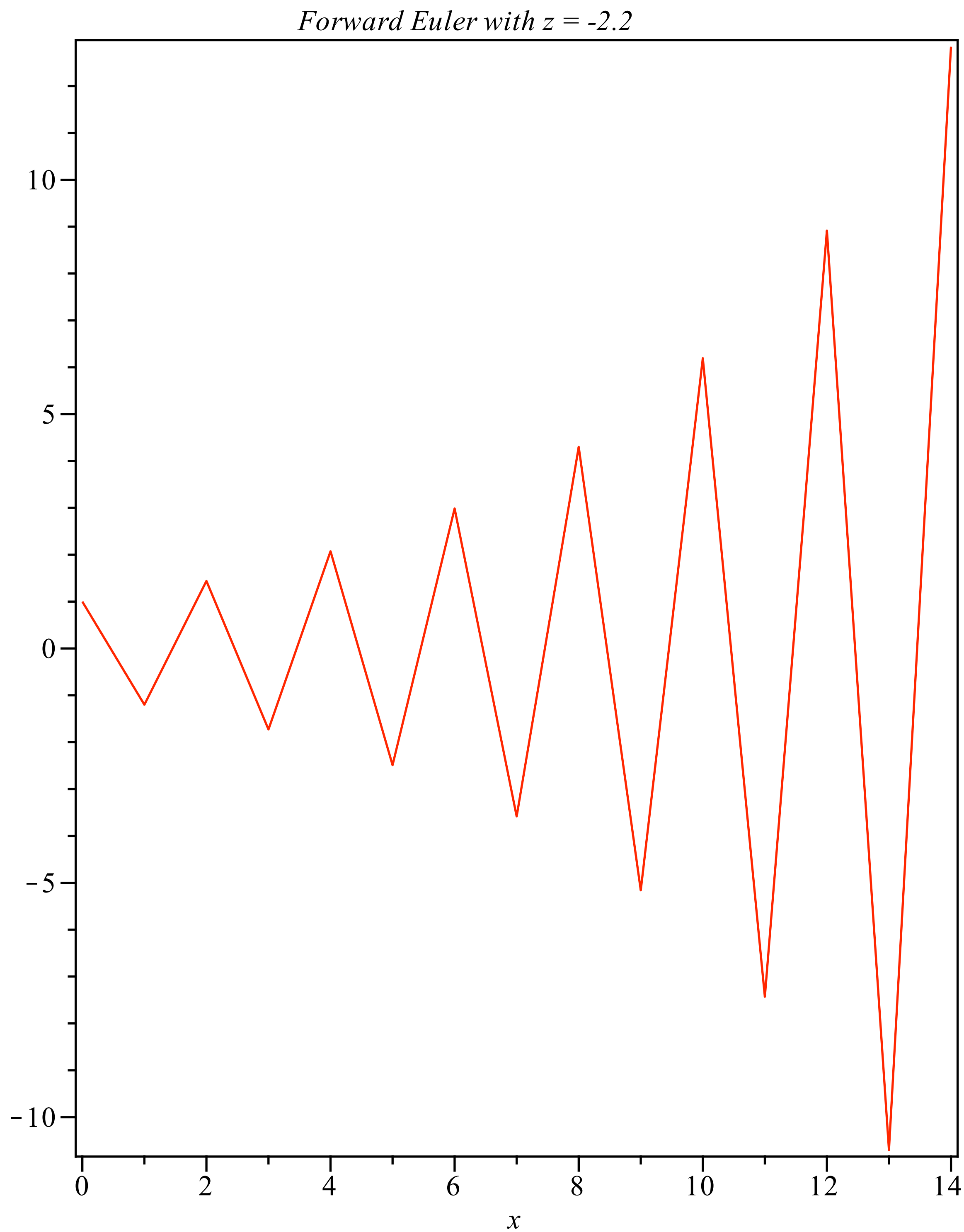
We know that the forward method is absolutely stable if -2 < z < 0. In this range (z<0), we expect to obtain a decaying exponential, which we do:

```
> plot(EULER(-0.1,forward),axes=boxed,labels=[`x`,``],title=
  `Forward Euler with z = -0.1`);
```



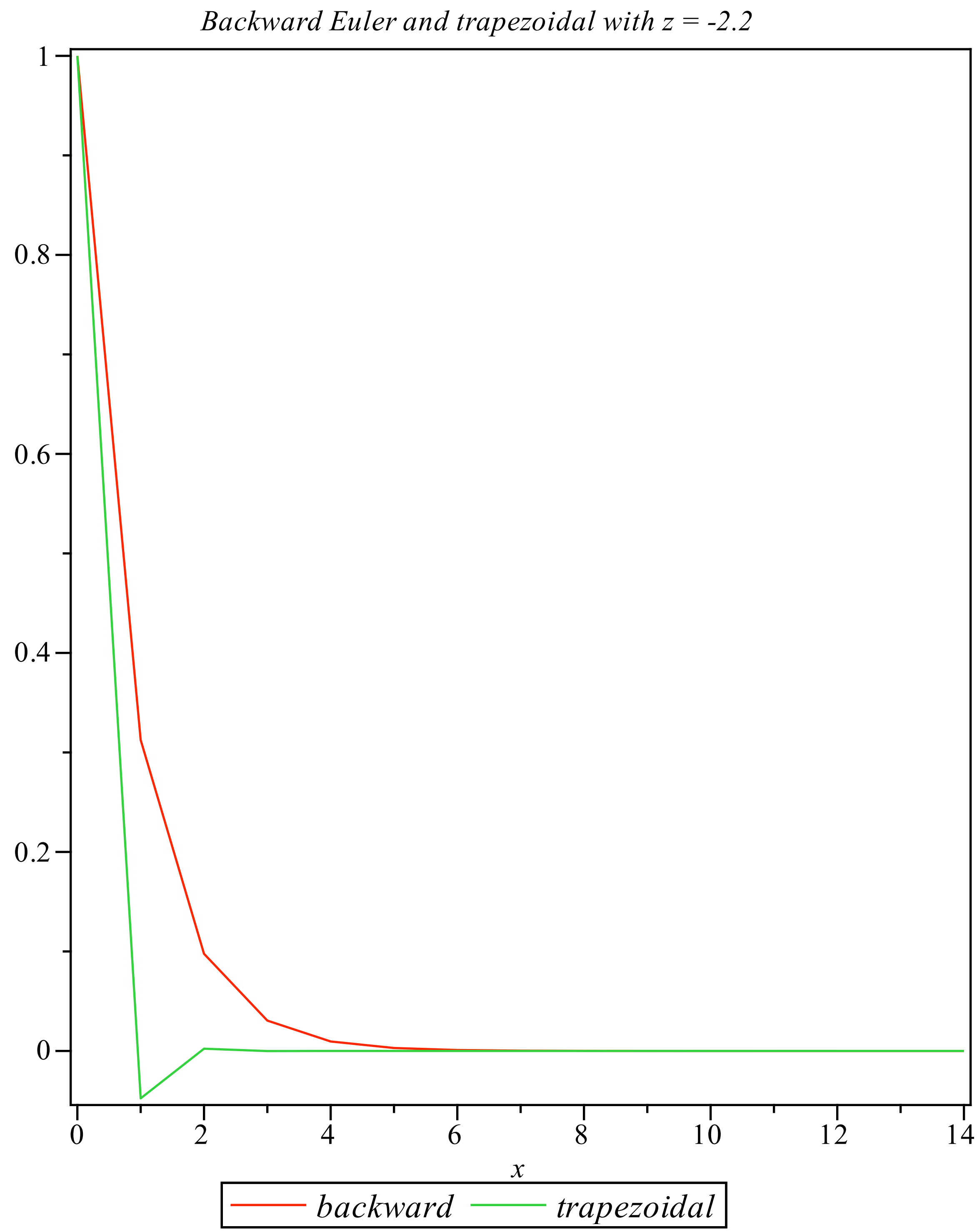
However, if z < -2, the method is unstable, which results in nothing like a decaying exponential. In fact the solution seems to grow exponentially:

```
> plot(EULER(-2.2,forward),axes=boxed,labels=[`x`,``],title=
  `Forward Euler with z = -2.2`);
```



But both the trapezoidal and backward methods are supposed to be stable for all z < 0; i.e. they give something that looks (sort-of) right:

```
> plot([EULER(-2.2,backward),EULER(-2.2,trap)],axes=boxed,labels=
  [`x`,``],title=`Backward Euler and trapezoidal with z = -2.2`,
  legend=[`backward`,`trapezoidal`]);
```



This all works in the complex plane too. Now if z is complex, y(x) will also be complex, so we define the following function that returns only the real part of the solution generated by EULER:

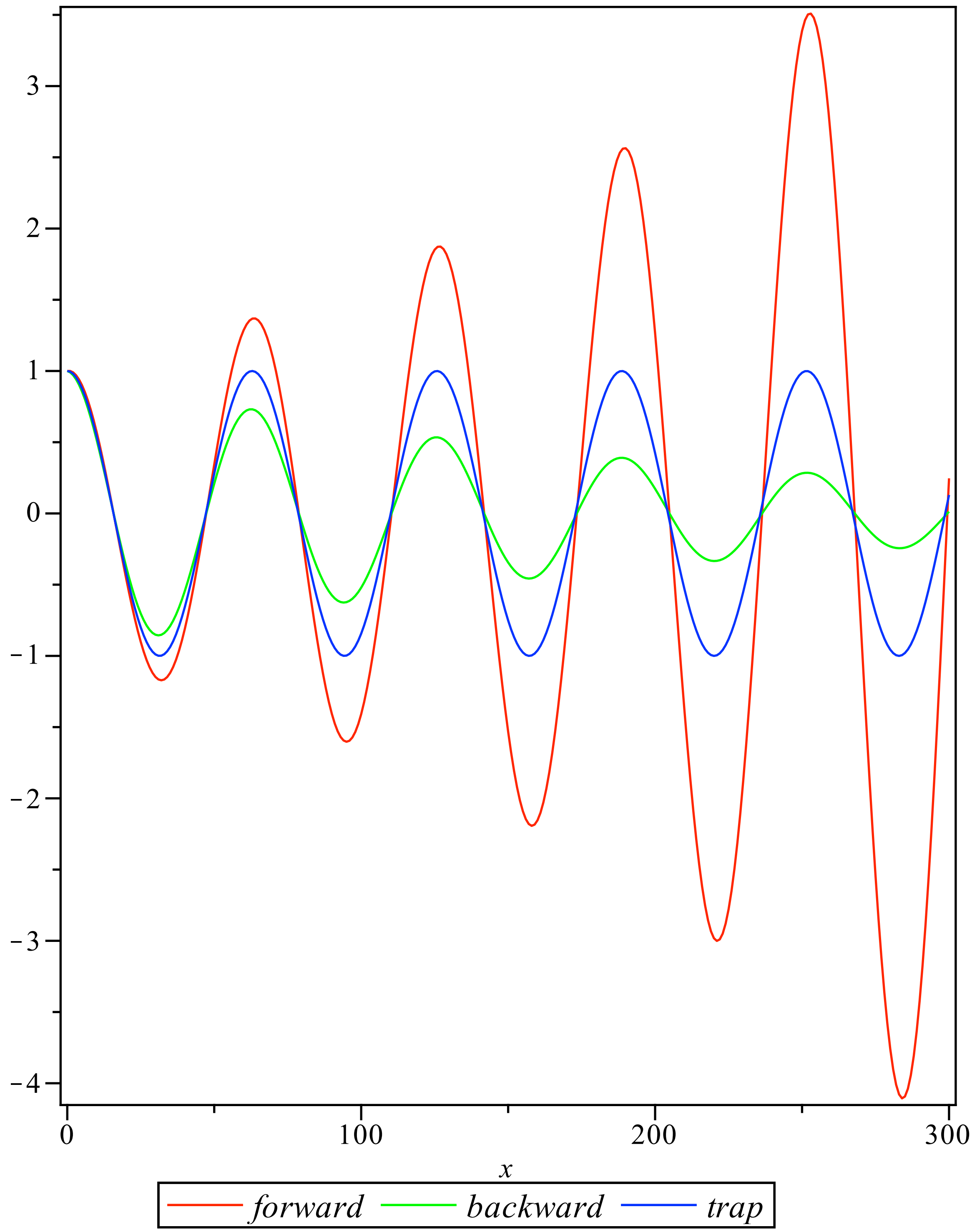
```
> REAL := proc(data)
        map(x -> [x[1],Re(x[2])],data):
  end proc:
```

Choosing z imaginary means that the forward Euler is unstable, while the other two methods are stable. The expected analytic solution in this case is a cosine function. The numeric solutions generated are:

```
> z := 0.1*I;
```

```
plot([REAL(EULER(z,forward)),REAL(EULER(z,backward)),REAL(EULER
(z,trap))],color=[red,green,blue],legend=[`forward`,`backward`,
`trap`],axes=boxed,labels=[`x`,``]);
```

$$z := 0.1\,\mathrm{I}$$



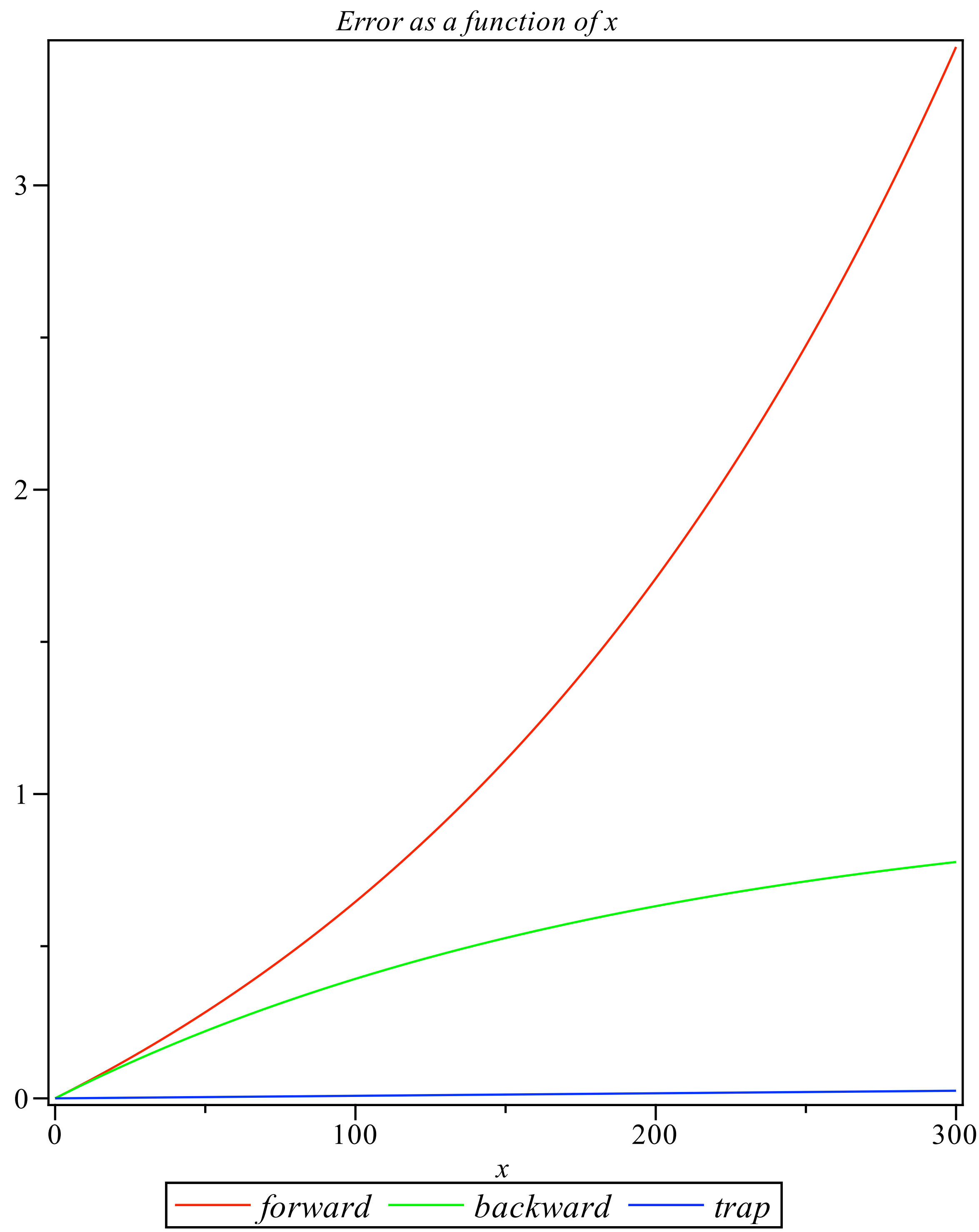
Note that the trap method actually does well reproducing the expected cosine-like behaviour. The forward solution seems to be exponentially growing in time (indicating the instability). While the backward solution does not blow-up (indicating it is stable), it doesn't do as good a job as the trap method. The following procedure calculates the error in the numeric approximation as a function of x.

```
> EULER_ERROR := proc(z,stencil)
        map(x->[x[1],abs(x[2]-exp(z*x[1]))],EULER(z,stencil)):
  end proc:
```

As can be seen in the below plot, the error in the forward stencil grow exponentially, while the error in the backward and trapezoidal stencil remains bounded (for z = 0.1*I). This is the hallmark of absolute stability: the fact that errors do not increase exponentially in time. Note that the errors for a stable method do not necessary decrease as the simulation continues, they just do not blow-up (as in the trap method).

```
> z := 0.1*I;
  plot([EULER_ERROR(z,forward),EULER_ERROR(z,backward),EULER_ERROR
  (z,trap)],color=[red,green,blue],legend=[`forward`,`backward`,
  `trap`],axes=boxed,labels=[`x`,``],title=`Error as a function of
  x`);
```

$$z := 0.1\,\mathrm{I}$$



Error as a function of x
3
2
1
0
0
100
200
300
x
forward
backward
trap

>