Floating Point Numbers Review

Reminder that **normalised** floating point numbers are stored as

$$(-1)^s \times 1.d_1 d_2 \ldots d_n \times 2^{\exp}$$

or equivalently

$$(-1)^s \times \left(1 + d_1 \cdot 2^{-1} + d_2 \cdot 2^{-2} + \ldots + d_n \cdot 2^{-n}\right) \times 2^{\exp}$$

where $s = 0, 1$ is the sign bit, $d_i = 0, 1$ are the mantissa bits and the exponent $\exp$ ranges from $\exp_{\min}$ to $\exp_{\max}$. Normalised means that $d_0 = 1$.

For example, the Float32 data structure is normalised and has 1 sign bit, 8 exponent bits, and 23 mantissa bits (also known as precision bits). The exponent is given as the number represented in the exponent bits - 127.

Question 1

What is $\frac{1}{2}$ in Float32?

Sign bit $s = 0$

$0.5 = 1.\underbrace{0000000}_{\text{mantissa}} \times 2^{-1}$ in binary

So $\exp = 127 - 1 = 126 = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 = 01111110$ in binary

So representation in Float32 = $\underbrace{0}_{\text{sign}}\ \underbrace{01111110}_{\text{exponent}}\ \underbrace{0000...000}_{\text{mantissa}}$

Question 2

What is 0.75 in Float32?

Sign bit $s = 0$

$0.75 = 0.5 + 0.25 = 1.\underbrace{100000}_{\text{mantissa}} \times 2^{-1}$ in binary

So $\exp = 127 - 1 = 126 = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 = 01111110$ in binary

So representation in Float32 = $\underbrace{0}_{\text{sign}}\ \underbrace{01111110}_{\text{exponent}}\ \underbrace{1000...000}_{\text{mantissa}}$

Question 3

What is the smallest (positive) integer not represented exactly using Float32?

Any positive power of 2 less than $2^{255-127=128}$ is represented exactly. This is because you can just set the mantissa to zero and the exponent power to be what you want.

This means that $2^{23}+1$ is the largest integer not represented exactly. Any integer less than 2^{23} can be represented exactly using the 23 mantissa bits, while 2^{23} cannot be it is a power of 2 so can be done also by setting the mantissa to zero and setting the correct exponent.

Question 4

What is the smallest Float32 number greater than 1?

1 in Float32 $= 1.\underbrace{000000}_{\text{mantissa}} \times 2^0$ in binary

So next biggest number is just increasing the mantissa by smallest thing possible $= 1.\underbrace{000000...01}_{\text{mantissa}} \times 2^0$ in binary $= 1 + 2^{-23}$

Differential Equations Review

The most basic case is solving a scalar ODE with initial condition $y(x_0) = y_0$

$$y'(x) = f\Big(x, y(x)\Big)$$

The first step is to apply the fundamental theorem of calculus and write this as an integral equation

$$y(x) = y_0 + \int_{x_0}^{x} y'(s)ds$$

Plug in the formula

$$y(x) = y_0 + \int_{x_0}^{x} f\Big(s, y(s)\Big)ds$$

Then the goal is to approximate the integral term using whatever method you like.

Question 1

Approximate the integral using a left Riemann sum. What is the resulting scheme?

This is Forward Euler. Approximate integral is

$$\int_{x_0}^{x} f\Big(s, y(s)\Big) ds \approx (x - x_0) f\Big(x_0, y_0\Big)$$

Which gives the scheme

$$y(x) = y_0 + (x - x_0) f\Big(x_0, y_0\Big)$$

Question 2

Approximate the integral using the trapezoidal rule. What is the resulting scheme?

Approximate integral is

$$\int_{x_0}^{x} f\Big(s, y(s)\Big) ds \approx (x - x_0)\Bigg(f\Big(x_0, y_0\Big) + f\Big(x, y(x)\Big)\Bigg) * 0.5$$

Which gives the scheme

$$y(x) = y_0 + (x - x_0)\Bigg(f\Big(x_0, y_0\Big) + f\Big(x, y(x)\Big)\Bigg) * 0.5$$

Question 3

Consider the case now where $f'(x) = 2f(x)$. Write code to find the solution at $x = 1$ using the trapezoidal rule. Use $\Delta x = 10^{-2}$ and initial condition $f(0) = 1$.

Plot the computed solution and the exact solution (you need to solve for the exact solution).

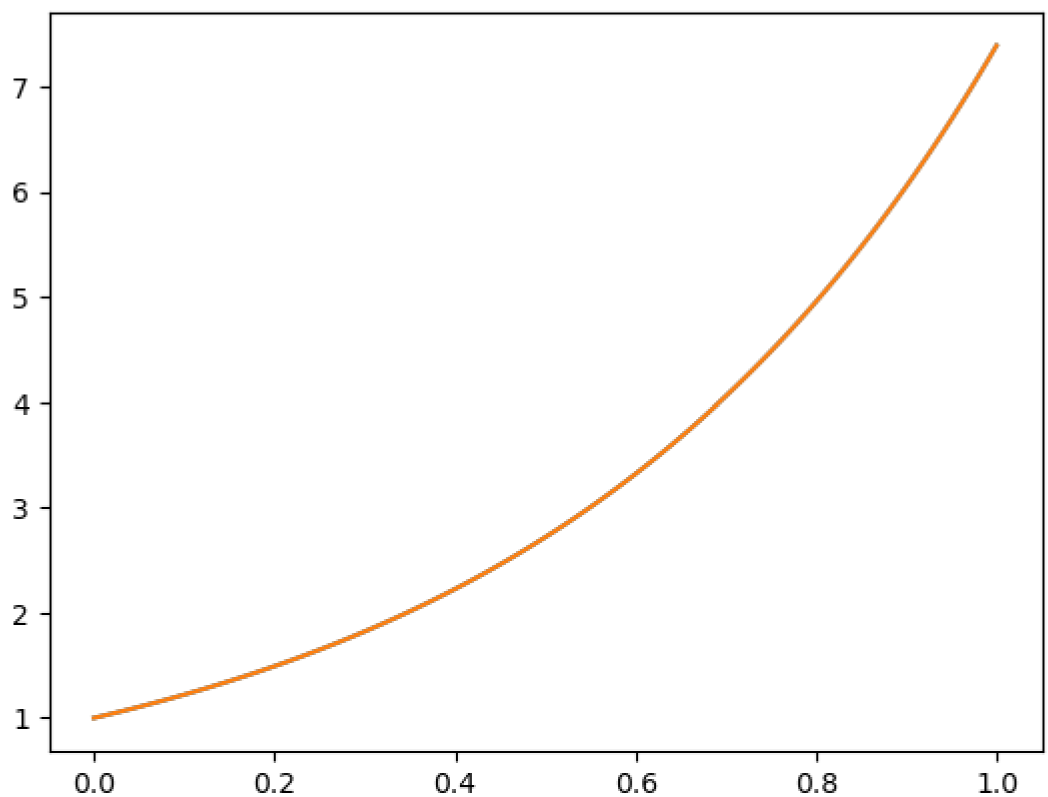
Plugging in to the equation above gives the scheme:

$$\begin{gathered}
y(x) = y_0 + (x - x_0)(2y_0 + 2y) * 0.5 \\
y(x) = y_0 + (x - x_0)y + (x - x_0)y_0 \\
(1 - x + x_0)y(x) = y_0 + (x - x_0)y_0 \\
y(x) = \frac{(1 + x - x_0)y_0}{1 - x + x_0}
\end{gathered}$$

In [1]:
```
using PyPlot

function trapezoidal()

    dx = 1e-2;
    N = round( 1/dx );
    xvals = [0.0];
    fvals = [1.0];
    x = 0.0;

    for ii = 1:N
        x += dx;
        push!( xvals, x );
        push!( fvals, ( fvals[end] * (1+dx) ) / (1-dx) );
    end

    plot( xvals, fvals )
    plot( xvals, [exp(2*x) for x in xvals] )

end
```

Out[1]:
```
trapezoidal (generic function with 1 method)
```

In [2]:
```
trapezoidal();
```



So we see the solution matches up extremely well.

Question 4

Consider now the problem of $f''(x) = -4f(x)$. Introducing a new variable $v(x) = f'(x)$ rewrite this as a system of first order equations.

$$v'(x) = -4f(x), f'(x) = v(x)$$

$$\underbrace{\begin{bmatrix} v(x) \\ f(x) \end{bmatrix}'}_{u'} = \underbrace{\begin{bmatrix} 0 & -4 \\ 1 & 0 \end{bmatrix}}_{A}\underbrace{\begin{bmatrix} v(x) \\ f(x) \end{bmatrix}}_{u}$$

Question 5

Code up the above using the trapezoidal rule. Set an initial condition of $u(0) = 0, u'(0) = 2$.

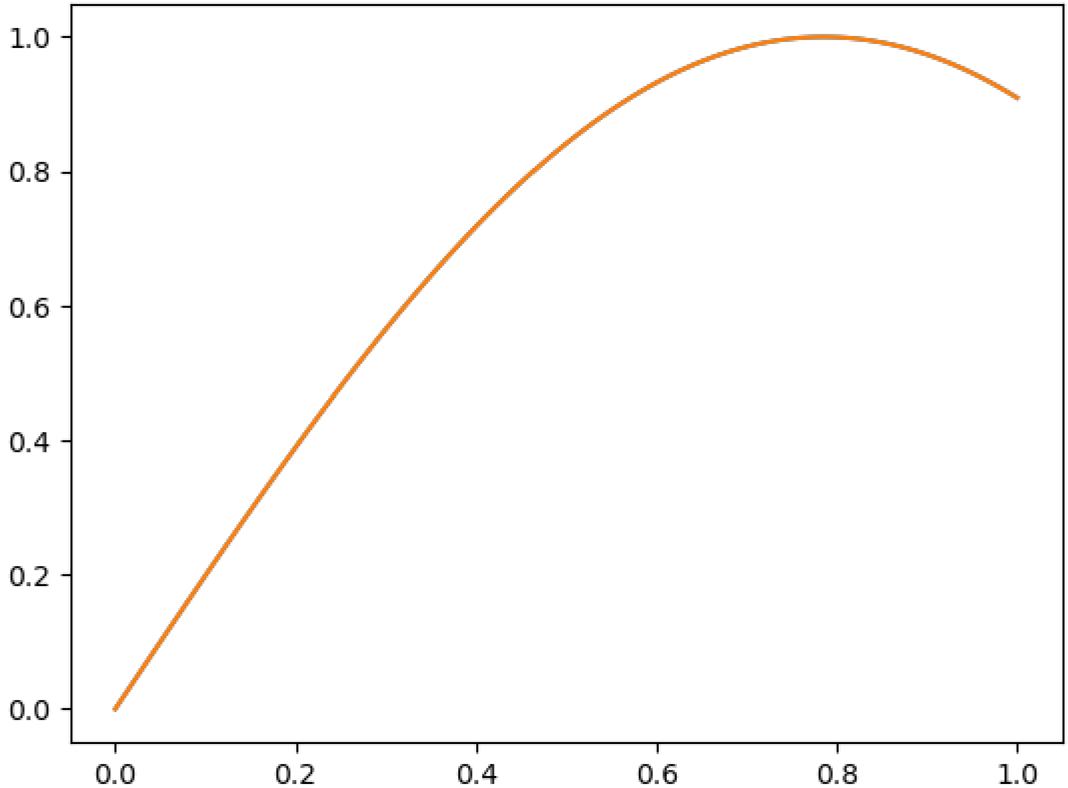
Plugging in to the equation above gives the scheme:

$$u(x) = u_0 + \frac{x - x_0}{2}\left(Au(0) + Au(x)\right)$$
$$u(x) = u_0 + \frac{x - x_0}{2}Au(0) + \frac{x - x_0}{2}Au(x)$$
$$(I - \frac{x - x_0}{2}A)u(x) = (I + \frac{x - x_0}{2}A)u_0$$
$$u(x) = \left(I - \frac{x - x_0}{2}A\right)^{-1}\left(I + \frac{x - x_0}{2}A\right)u_0$$

In [5]:
```
using LinearAlgebra

function trapezoidalsystem()

    dx = 1e-2;
    N = round( 1/dx );
    xvals = [0.0];
    fvals = [ [2.0,0.0] ];
    x = 0.0;

    A = [ [0,1] [-4,0] ]

    for ii = 1:N
        x += dx;
        push!( xvals, x );
        push!( fvals, ( I-dx*A*0.5 ) \ ( (I+dx*A*0.5)*fvals[end] ) );
    end

    plot( xvals, [ fval[2] for fval in fvals ] )
    plot( xvals, [ sin(2*x) for x in xvals] )

end
```

Out[5]:
```
trapezoidalsystem (generic function with 1 method)
```

In [6]:
```
trapezoidalsystem();
```



So the solution also matches well here.

In [ ]:
```
```