# Assignment 7: Euler's Method and Backward Euler's

By : Abrar Imon

Part 1 : First, I will create a function called eulerMethod that will take as input, the differential equation, the initial t value, the y( initial t), the step size and the t for which we want to estimate y(t). Then, this function will perform however many iterations of the Euler's method it needs to estimate y(t) and then the function will return this estimate.

In[ ]:=
```
eulerMethod[diffEq_, initialT_, initialY_, stepSize_, finalT_] := (
  t = initialT;
  w = initialY;
  totalSteps = (finalT - initialT) / stepSize;
  For[i = 0, i < totalSteps, i++, w = N[w + stepSize * diffEq[t, w], 32];
   t = t + stepSize];
  Return[N[w, 5]]
 )
```

In[ ]:=
```
(*This cell sets up the differential equation for part 1.*)
yPrime[t_, y_] := 1 - t + 4 y
```

In[ ]:=
```
(*This cell sets up the solution to the above differential equation
  with initial condition y(0)=1. This is the same solution we found
  in class. We will need this when we are comparing the Euler's method
  and backward Euler's estimate with the exact analytical solution.*)
y[t_] := N[t / 4 - 3 / 16 + (19 / 16) E^(4 t), 5]

(*Part 1.a: This cell makes a grid to show our estimations for y(t) for
   different values of t and different step sizes using the Euler's method.*)
Grid[Table[{t, eulerMethod[yPrime, 0, 1, 0.05, t], eulerMethod[yPrime, 0, 1, 0.025, t],
   eulerMethod[yPrime, 0, 1, 0.01, t], eulerMethod[yPrime, 0, 1, 0.001, t], y[t]},
  {t, {0, 0.1, 0.2, 0.5, 1, 1.5, 2}}], Frame → All, ItemSize → 7]
```

```
(*This cell titles the columns of the grid.*)
ReplacePart[%64, 1 → Prepend[First[%64],
   {"t", "h = 0.05", "h = 0.025", "h = 0.01", "h = 0.001", "Exact Value"}]]
```

Out[ ]=

| t | h = 0.05 | h = 0.025 | h = 0.01 | h = 0.001 | Exact Value |
|---|---|---|---|---|---|
| 0 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| 0.1 | 1.5475 | 1.57612 | 1.59529 | 1.60763 | 1.60904 |
| 0.2 | 2.3249 | 2.40801 | 2.46446 | 2.50112 | 2.50533 |
| 0.5 | 7.29019 | 7.92641 | 8.37669 | 8.67707 | 8.712 |
| 1 | 45.5884 | 53.8079 | 60.0371 | 64.3826 | 64.8978 |
| 1.5 | 282.072 | 361.759 | 426.408 | 473.56 | 479.259 |
| 2 | 1745.67 | 2432.79 | 3029.33 | 3484.16 | 3540.2 |

In[ ]:=
```
(*Now, I will create a function for backward
 Euler's. This is similar to the first function,
but in the formula for calculating "w", in the input for the differential equation,
I replaced t with (t+stepSize) and I replaced w with y[t+stepSize]. This is the
 same formula we were given in class for the for backward Euler's, in class.*)
backwardEuler[diffEq_, initialT_, initialY_, stepSize_, finalT_] := (
  t = initialT;
  w = initialY;
  totalSteps = (finalT - initialT) / stepSize; For[i = 0, i < totalSteps,
   i++, w = N[w + stepSize * diffEq[t + stepSize, y[t + stepSize]], 32];
   t = t + stepSize];
  Return[N[w, 5]]
 )

(*Part 1.b: This cell makes a grid to show our estimations for y(t) for
   different values of t and different step sizes using the backward Euler's.*)
Grid[Table[{t, backwardEuler[yPrime, 0, 1, 0.05, t], backwardEuler[yPrime, 0, 1, 0.025, t],
   backwardEuler[yPrime, 0, 1, 0.01, t], backwardEuler[yPrime, 0, 1, 0.001, t], y[t]},
  {t, {0, 0.1, 0.2, 0.5, 1, 1.5, 2}}], Frame → All, ItemSize → 7]

(*This cell titles the columns of the grid.*)
ReplacePart[%70, 1 → Prepend[First[%70],
   {"t", "h = 0.05", "h = 0.025", "h = 0.01", "h = 0.001", "Exact Value"}]]
```

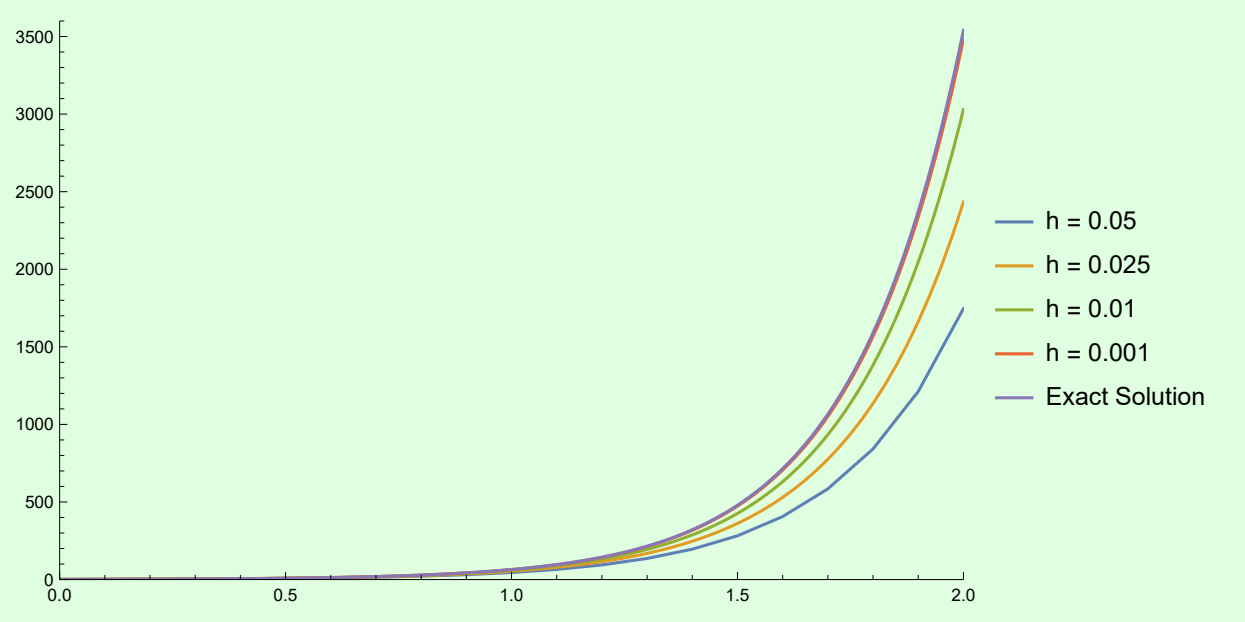
Out[ ]=

| t | h = 0.05 | h = 0.025 | h = 0.01 | h = 0.001 | Exact Value |
|---|---|---|---|---|---|
| 0 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| 0.1 | 1.66939 | 1.63873 | 1.6208 | 1.61021 | 1.60904 |
| 0.2 | 2.65571 | 2.57931 | 2.53463 | 2.50824 | 2.50533 |
| 0.5 | 9.49598 | 9.09768 | 8.86476 | 8.72719 | 8.712 |
| 1 | 71.4746 | 68.1332 | 66.1792 | 65.0252 | 64.8978 |
| 1.5 | 528.639 | 503.552 | 488.881 | 480.216 | 479.259 |
| 2 | 3905.86 | 3720.08 | 3611.45 | 3547.28 | 3540.2 |

In[31]:=
```
(*Part 1.c: This cell makes lists with the eulerMethod function with different step
   sizes. The list will be in a form that we can easily plot, using ListLinePlot[].*)
eulerStepSizePoint05 = Table[{t, eulerMethod[yPrime, 0, 1, .05, t]}, {t, 0, 2, 0.05}];
eulerStepSizePoint025 = Table[{t, eulerMethod[yPrime, 0, 1, .025, t]}, {t, 0, 2, 0.025}];
eulerStepSizePoint01 = Table[{t, eulerMethod[yPrime, 0, 1, .01, t]}, {t, 0, 2, 0.01}];
eulerStepSizePoint001 = Table[{t, eulerMethod[yPrime, 0, 1, .001, t]}, {t, 0, 2, 0.001}];
exactValues = Table[{t, y[t]}, {t, 0, 2, 0.0001}];

(*This cell plots the exact solution and also plots
 our Euler's method estimate with the different step sizes.*)
ListLinePlot[{eulerStepSizePoint05, eulerStepSizePoint025, eulerStepSizePoint01,
  eulerStepSizePoint001, exactValues}, PlotRange → {{0, 2}, {0, 3600}}, PlotLegends →
  {"h = 0.05", "h = 0.025", "h = 0.01", "h = 0.001", "Exact Solution"}, ImageSize → 500]
```

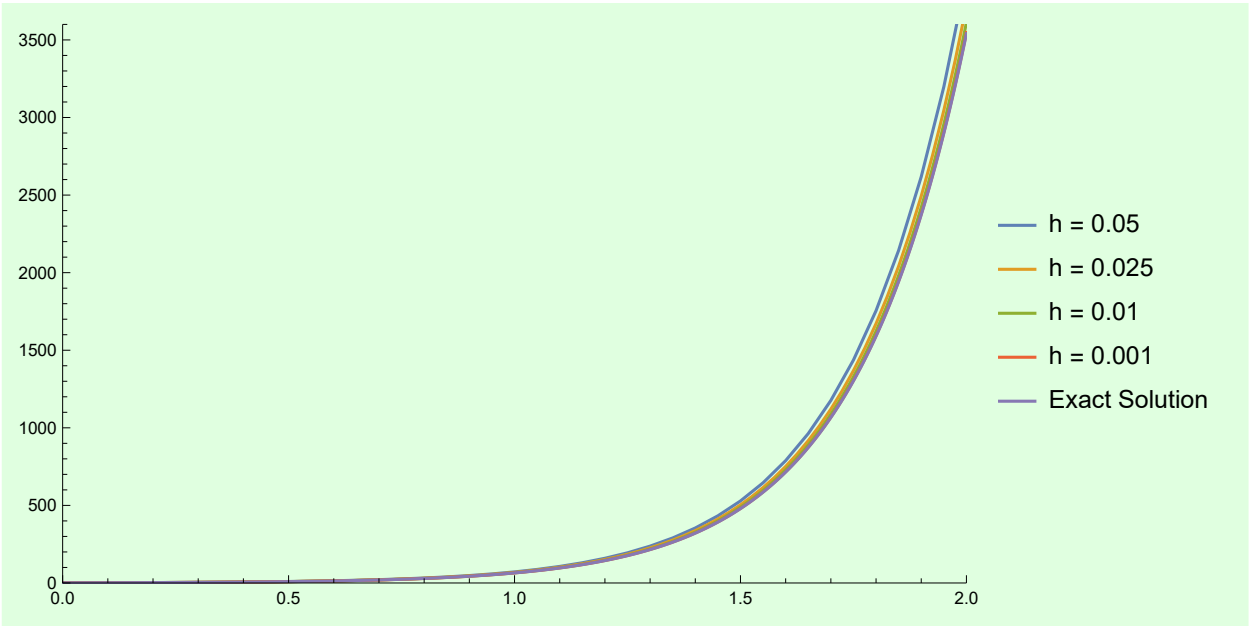
Out[•]=




```
(*Part 1.d: This cell makes lists with the backwardEuler function with different step
   sizes. The list will be in a form that we can easily plot, using ListLinePlot[].*)
backwardEulerStepSizePoint05 =
 Table[{t, backwardEuler[yPrime, 0, 1, .05, t]}, {t, 0, 2, 0.05}];
backwardEulerStepSizePoint025 =
 Table[{t, backwardEuler[yPrime, 0, 1, .025, t]}, {t, 0, 2, 0.025}];
backwardEulerStepSizePoint01 =
 Table[{t, backwardEuler[yPrime, 0, 1, .01, t]}, {t, 0, 2, 0.01}];
backwardEulerStepSizePoint001 =
 Table[{t, backwardEuler[yPrime, 0, 1, .001, t]}, {t, 0, 2, 0.001}];
```

In[ ]:=
```
(*This cell plots the exact solution and also plots
 our backward Euler's estimate with the different step sizes.*)
ListLinePlot[{backwardEulerStepSizePoint05, backwardEulerStepSizePoint025,
  backwardEulerStepSizePoint01, backwardEulerStepSizePoint001, exactValues},
 PlotRange → {{0, 2}, {0, 3600}}, PlotLegends →
  {"h = 0.05", "h = 0.025", "h = 0.01", "h = 0.001", "Exact Solution" }, ImageSize → 500]
```

Out[ ]=




Part 2: First, I will create a function to perform the Euler's method for the new equation. This equation will be different from the original one, because it takes in as input the number of steps and computes the h value from that. Then, it outputs its estimate for u(T).

```
eulerMethodNewEquation[diffEq_, leftBound_, rightBound_, numberOfSteps_, T_] :=
 ( stepSize = ((rightBound - leftBound) / numberOfSteps);
  t = leftBound;
  w = 2;
  totalSteps = (T - leftBound) / stepSize;
  For[i = 0, i < totalSteps, i++, w = N[w + stepSize * diffEq[t, w], 32];
   t = t + stepSize];
  Return[N[w, 5]]
 )
```

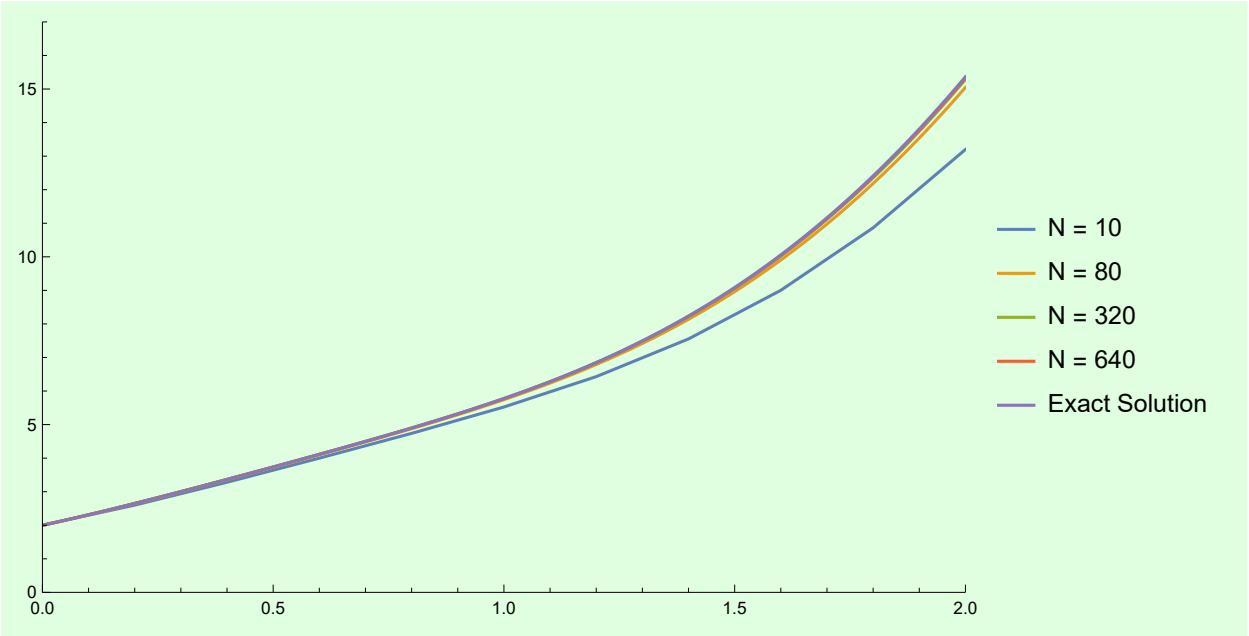
In[86]:=
```
(*This cell sets up the differential equation for part 2.*)
uPrime[t_, u_] := Cos[Pi * t] + u
```

In[85]:=
```
(*This cell sets up the analytical solution to the initial problem for part 2.*)
u[t_] := ((3 E^t) + (2 (Pi^2) E^t) - Cos[Pi * t] + Pi * Sin[Pi * t]) / (1 + Pi^2)
```

```
(*Part 2.a: This cell makes lists with the
   eulerMethodNewEquation[] function with different number of steps. The
   list will be in a form that we can easily plot, using ListLinePlot[].*)
forwardEulerNis10 = Table[{t, eulerMethodNewEquation[uPrime, 0, 2, 10, t]},
  {t, 0, 2, 2 / 10}]; forwardEulerNis80 =
 Table[{t, eulerMethodNewEquation[uPrime, 0, 2, 80, t]}, {t, 0, 2, 2 / 80}];
forwardEulerNis320 =
 Table[{t, eulerMethodNewEquation[uPrime, 0, 2, 320, t]}, {t, 0, 2, 2 / 320}];
forwardEulerNis640 =
 Table[{t, eulerMethodNewEquation[uPrime, 0, 2, 640, t]}, {t, 0, 2, 2 / 640}];
newExactValues = Table[{t, u[t]}, {t, 0, 2, 0.0001}]
```

```
(*This cell plots the exact solution and also plots our
 Euler's method estimates with the different numbers of steps.*)
ListLinePlot[{forwardEulerNis10, forwardEulerNis80, forwardEulerNis320,
  forwardEulerNis640, newExactValues}, PlotRange → {{0, 2}, {0, 17}}, PlotLegends →
  {"N = 10", "N = 80", "N = 320", "N = 640", "Exact Solution" }, ImageSize → 500]
```

Out[51]=

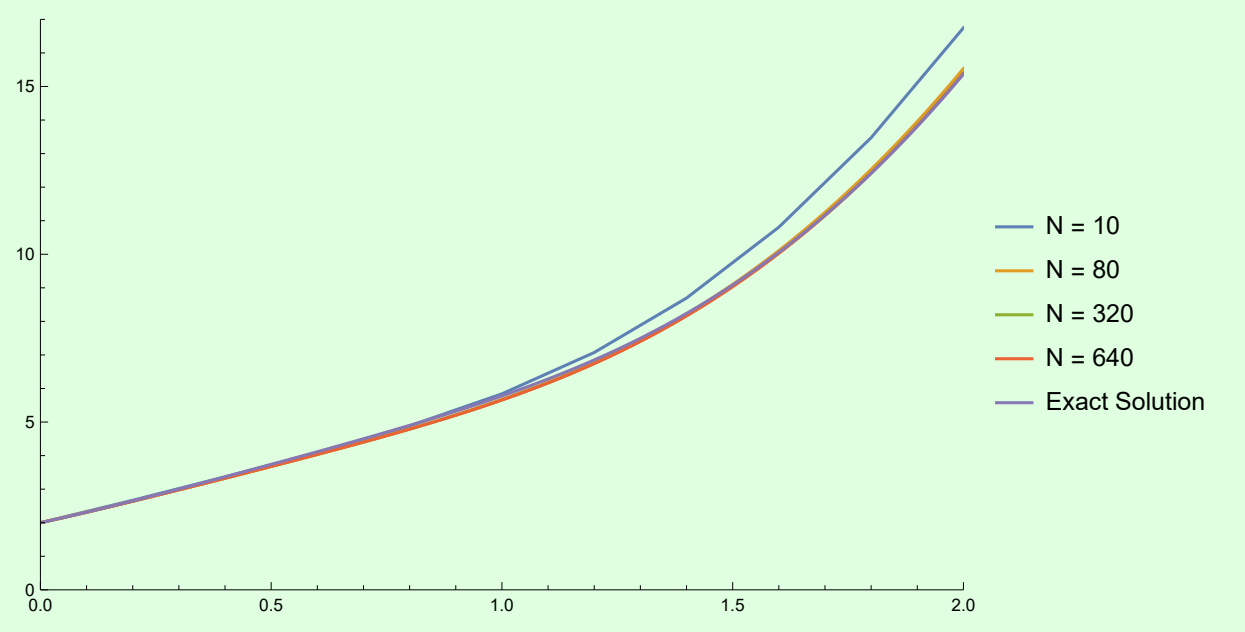



```
(*This is similar to the last function, but the formula
 (to calculate w in each iteration) is changed to the one for the backward Euler's.*)
backwardEulerMethodNewEquation[diffEq_, leftBound_, rightBound_, numberOfSteps_, T_] :=
 ( stepSize = ((rightBound - leftBound) / numberOfSteps);
  t = leftBound;
  w = 2;
  totalSteps = (T - leftBound) / stepSize; For[i = 0, i < totalSteps,
   i++, w = N[w + stepSize * diffEq[t + stepSize, u[t + stepSize]], 32];
   t = t + stepSize];
  Return[N[w, 5]]
 )
```

```
(*Part 2.b: This cell makes lists with the backwardEulerMethodNewEquation[]
   function with different numbers of steps. The list will be
   in a form that we can easily plot, using ListLinePlot[].*)
backwardEulerNis10 = Table[{t, backwardEulerMethodNewEquation[uPrime, 0, 2, 10, t]},
  {t, 0, 2, 2 / 10}]; backwardEulerNis80 =
 Table[{t, backwardEulerMethodNewEquation[uPrime, 0, 2, 80, t]}, {t, 0, 2, 2 / 80}];
backwardEulerNis320 =
 Table[{t, backwardEulerMethodNewEquation[uPrime, 0, 2, 320, t]}, {t, 0, 2, 2 / 320}];
backwardEulerNis640 =
 Table[{t, backwardEulerMethodNewEquation[uPrime, 0, 2, 640, t]}, {t, 0, 2, 2 / 640}];

(*This cell plots the exact solution and also plots our backward
 Euler's method estimates with the different numbers of steps.*)
ListLinePlot[{backwardEulerNis10, backwardEulerNis80, backwardEulerNis320,
  backwardEulerNis640, newExactValues}, PlotRange → {{0, 2}, {0, 17}}, PlotLegends →
  {"N = 10", "N = 80", "N = 320", "N = 640", "Exact Solution"}, ImageSize → 500]
```

Out[52]=



In[88]:=
```
(*Part 2.
 c:This Grid[] function below will show the max error for the forward Euler's and
   backward Euler's for different values of n.To get the max error for a particular n,
inside the Grid[] function is a Max[] function in which there is a list,
made up of the differences between Euler's
 estimate and the actual solution at different t values.*)
Grid[Join[{{"Number of steps (n)",
    "Max Error Forward Euler's", "Max Error Backward Euler's"}}, Table[
   {n, Max[Table[Abs[u[t] - eulerMethodNewEquation[uPrime, 0, 2, n, t]], {t, 0, 2, 2 / n}]],
    Max[Table[Abs[u[t] - backwardEulerMethodNewEquation[uPrime, 0, 2, n, t]],
      {t, 0, 2, 2 / n}]]}, {n, {10, 80, 320, 640}}]], Frame → All]
```

Out[88]=

| Number of steps (n) | Max Error Forward Euler's | Max Error Backward Euler's |
|---|---|---|
| 10 | 2.168 | 1.381 |
| 80 | 0.319 | 0.168 |
| 320 | 0.081 | 0.042 |
| 640 | 0.041 | 0.021 |

As n increases, the max error decreases for both methods. If we round a little bit, when n doubled from 320 to 640, the max error got cut in half for both of the methods. When n quadrupled from 80 to 320, the max error was divided by 4 for both methods.