Daniel Reid, BBM iteration assignment

1 Background

The purpose of this assignment is to compare the leapfrog and predictor-corrector schemes in modeling a low amplitude wave using the Benjamin-Bona-Mahony (BBM) partial differential equation, and to describe the model's accuracy and computation time with respect to time and position step size. The BBM equation is shown in equation 1.

$$u_t + ux + uu_x - u_{xxt} = 0 \tag{1}$$

The time derivative can be isolated as shown in Equation 2.

$$u_t = -(1 - \frac{d^2}{dx^2})^{-1} \frac{d}{dx}(u + u^2) = B(u + u^2) \tag{2}$$

Equation 2 can be iterated using matrices to describe $\frac{d}{dx}(u + u^2)$ and $(1 - \frac{d^2}{dx^2})^{-1}$.

A center differencing scheme is used to take the derivative of $u + u^2$. The matrix populates $\frac{1}{2dx}$ on the upper diagonal and $\frac{-1}{2dx}$ on the lower diagonal.

The result of the matrix operating on $u + u^2$ is shown in Equation 3.

$$\frac{d(u + u^2)_i}{dx} = \frac{(u + u^2)_{i+1} - (u + u^2)_{i-1}}{2dx} \tag{3}$$

The matrix describing $1 - \frac{d^2}{dx^2}$ populates the upper and lower diagonals with $\frac{-1}{dx^2}$, and the main diagonal with $1 + \frac{2}{dx^2}$.

The iterating loop for the predictor corrector methods is shown below,

```
def iteratePredCor(nts, yuMtx, dt, invSvm, fod):
   for t in range(nts):
       prediction = yuMtx + eulerStep(dt, invSvm, fod, yuMtx)
       yuMtx = .5 * (prediction + yuMtx + eulerStep(dt, invSvm, fod, prediction))
return yuMtx
```

where eulerStep is defined as

```
def eulerStep(dt, invSvm, fod, yuMtx):
    return -dt * (invSvm * (fod * (yuMtx + sqrVals(yuMtx))))
```

The iterating loop for the leapfrog method is shown below.

```
def iterateLeapFrog(nts, yuMtx, dt, invSvm, fod):
    yuLast = yuMtx.copy()
    yuMtx = iteratePredCor(1, yuMtx, dt, invSvm, fod)
    nts-=1
```

```
    for t in range(nts):
        store = yuMtx.copy()
        yuMtx = yuLast + 2 * eulerStep(dt, invSvm, fod, yuMtx)
        yuLast = store
return yuMtx
```

The leapfrog method must store a copy of the last step, so two copies of the height list must be maintained.

The BBM equation can be integrated into equation 4

$$u(x,t) = \frac{3}{2}asech^2(\frac{1}{2}\sqrt{\frac{a}{a+1}}(x-(1+a)t)) \tag{4}$$

where a is the amplitude of the wave. This is used to evaluate the results of the iterations.

2 Results

The results are shown in the table below.

	Relative max error	dx	dt	nts	seconds
Pred/cor	1.93E-3	0.2	0.04	600	9
	6.40E-5	0.1	0.04	600	19
	3.40E-4	0.05	0.04	600	41
Pred/cor	6.40E-5	0.1	0.04	600	19
	4.70E-4	0.1	0.02	1200	38
	5.60E-4	0.1	0.01	2400	76
Leapfrog	2.01-3	0.2	0.04	600	5
	8.60-5	0.1	0.04	600	9
	3.36-4	0.05	0.04	600	21
Leapfrog	8.60E-5	0.1	0.04	600	9
	4.61E-4	0.1	0.02	1200	19
	5.56E-4	0.1	0.01	2400	38

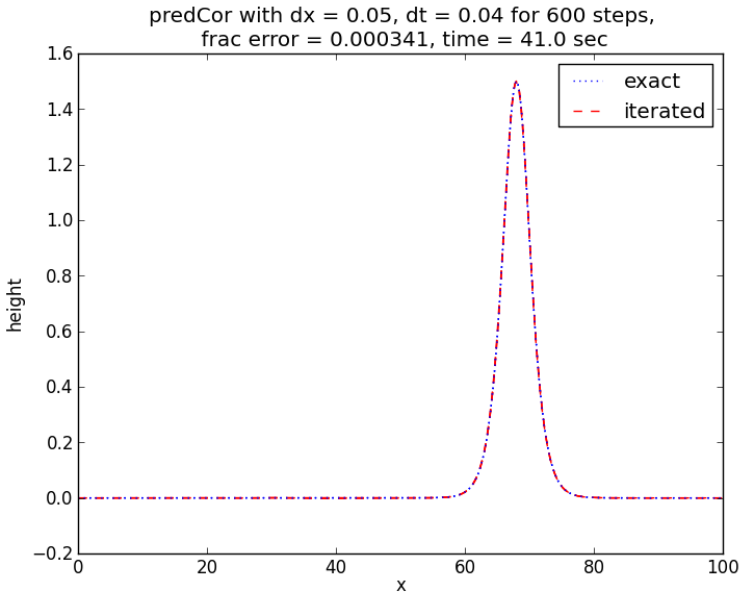
The results show that computation time scales linearly with number of time steps, which is expected since increasing the number of time steps does not increase the complexity of each time step. Computation time also scales roughly linearly with position step size, which is unexpected because the time to solve the next step should scale roughly with the number of elements in svm matrix, $\propto \frac{1}{dx^2}$. The lowest error was not at the smallest length step size. A small step size would give a large difference in the order or magnitude between the values in the fod and svm^{-1} matrices, which could introduce numerical error. The leapfrog method runs roughly twice as quickly as the predictor-corrector method, since a predictor-corrector is roughly twice as complex. The accuracy of the leapfrog method is roughly the same as that of the predictor-corrector method. It is possible they would diverge more in longer simulations, but inverting matrices for longer waves took too long. The relative max error was calculated using Equation 5,

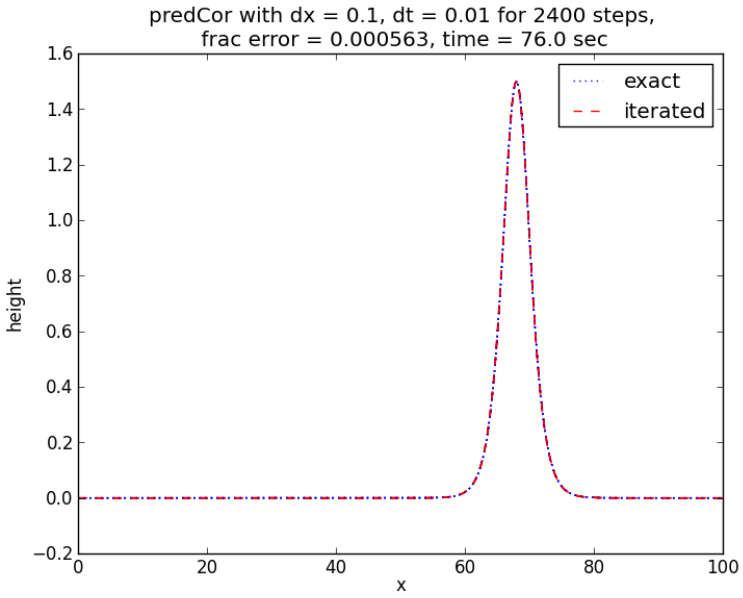
$$maxErr = \frac{abs(a[i]-b[i])}{avg(a[i],b[i])} \tag{5}$$

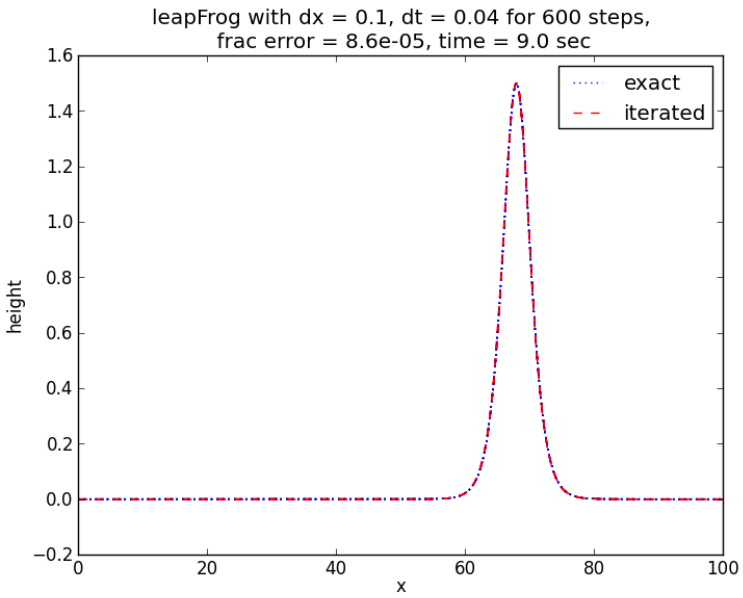
where a is the exact height list, b is the iterated height list, and i is the index of a's maximum.

The run times in this simulation are much slower than those in the class example implemented in Octave. This is probably Octave uses native code for its matrix operations while the libraries used here are implemented in Python. These Python libraries are likely slower since they are interpreted rather than compiled.

Some sample graphs are shown below.






leapFrog with dx = 0.1, dt = 0.04 for 600 steps,
frac error = 8.6e-05, time = 9.0 sec
exact
iterated
height
x
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
20
40
60
80
100