Research Article

On the Use of Backward Difference Formulae to Improve the Prediction of Direction in Market Related Data

E. Momoniat, 1 C. Harley, 1 and M. Berman 2

1 Centre for Differential Equations, Continuum Mechanics and Applications, School of Computational and Applied Mathematics, University of the Witwatersrand, Private Bag 3, Wits 2050, Johannesburg, South Africa
2 RAPA Pty Ltd., Level 10, 2 Bligh Street, Sydney, NSW 2000, Australia

Correspondence should be addressed to E. Momoniat; ebrahim.momoniat@wits.ac.za

Received 10 August 2013; Revised 13 October 2013; Accepted 14 October 2013

Academic Editor: Fenghua Wen

Copyright © 2013 E. Momoniat et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The use of a BDF method as a tool to correct the direction of predictions made using curve fitting techniques is investigated. Random data is generated in such a fashion that it has the same properties as the data we are modelling. The data is assumed to have “memory” such that certain information imbedded in the data will remain within a certain range of points. Data within this period where “memory” exists—say at time steps $t_1, t_2, \ldots, t_n$—is curve-fitted to produce a prediction at the next discrete time step $t_{n+1}$. In this manner a vector of predictions is generated and converted into a discrete ordinary differential representing the gradient of the data.

The BDF method implemented with this lower order approximation is used as a means of improving upon the direction of the generated predictions. The use of the BDF method in this manner improves the prediction of the direction of the time series by approximately 30%.

1. Introduction

In this brief note we show how a BDF method can be used as a corrector for predictions. BDF methods are backward differentiation formulae which are a family of multistep implicit methods. They are designed to solve initial value ordinary differential equations. The derivative of a function is approximated using information computed from earlier time steps, thereby increasing the accuracy of the approximation. This characteristic makes BDF methods ideal for our purposes where we seek to improve upon already existing data in the form of predictions. The predictions aim at accurately reflecting the direction of random data and are made using curve fitting techniques. This research comes out of a project undertaken to predict the direction of a subset of the South African market. The approach taken in analysing the data assumes that the data has a “memory.” More precisely this presupposes that a time series will have certain periods when the data has the same inherent information and dynamics. This allows us to conclude that the same information embedded in a previous selection of data points is still contained within the data point to be predicted from that set. While the data we generate in this paper is random with zero mean we are still able to show how an application of a BDF method improves the degree of accuracy in predicting the direction the data takes. The BDF formulae are constructed by satisfying the differential equation exactly at one point $t_n$ and then interpolating $k$ previous points. A Lagrangian interpolation is typically used. The initial prediction of direction is made using linear/spline curve fitting. The implementation of the BDF is not done directly; rather it is combined with a lower order approximation of the gradient of the data vector which leads to the difference equation we aim to use. The novelty of the approach taken here is that we iterate the difference equation structured from the BDF formula and the lower order approximation of the gradient to convergence.

The random walk hypothesis has had its fair share of attention as a means of explaining stock price movements. This financial theory states that stock market prices evolve according to a random walk and thus the prices of the stock market cannot be predicted. While the work undertaken in this paper concurs with this theory with regard to the random walk followed by the relevant data, we still maintain an assumption that there exists embedded information within
2. Algorithm Description

The first part of this analysis is to generate random data that may be used to simulate an actual financial time series. Here we use the MATLAB function `randn` that generates pseudorandom scalar values drawn from a normal distribution with mean zero and standard deviation one. The code used to generate this data is presented in Algorithm 1. We start out with an element that has value zero and then start stepping along the x-axis. The direction in which we step is dependent on whether the number outputted by `randn` is greater or less than a half. The magnitude of the step is determined by the magnitude of the number outputted by `randn`. We continue stepping in this way a finite number of times; 100 steps were chosen in this instance. The last data point is put into a vector which we use to simulate our time series of returns. Only the last point is chosen since the vector which has just been created consists of points with relative small errors between them; that is, this vector is not a good representation of the data we are trying to simulate. We continue running the random walk until we have a vector of returns. We subtract the mean and divide by the variance in order to produce data that is more reflective of financial returns obtained from price movements, that is, data within the range [0, 1]. The return data generated in this way has zero mean with standard deviation smaller than one. The standard deviation is not one since the data has not been normalised; that is, we divided by the variance and not the standard deviation. This was done in order to maintain the strict range of [0, 1] for the data. In Figure 1 we plot three simulations of return data obtained in this way. When we compare the generated data to an actual times series of returns from the South African market we find that our simulation is fairly accurate since the movements exhibited by the returns of the South African market are similar to those depicted in Figure 1.

We then choose an appropriate length of data to indicate what we term “memory” in the data. For the purposes of this paper we assume that using four initial data points is sufficient to account for the memory. Given the fact that the behaviour observed in Figure 1 is highly oscillatory a short memory span of four points seems sensible for a data series of 100 points as was chosen here. This indicates that any rational underlying dynamics are not observed in the long term but rather in the short term. We then curve-fit through the initial four points of the vector which represents our actual known data points, \( y_1, y_2, y_3, \) and \( y_4 \). We use this fitted curve to predict the value of the fifth point, \( y_5 \). We then use data values two to five, \( y_2, y_3, y_4, \) and \( y_5 \), to predict the sixth value, \( y_6 \). Continuing in this way we end up with a vector of actual known values and
We are incorporating the BDF method in a direct fashion. We are incorporating ordinary differential equation values. Some examples of BDF methods for solving the first order ordinary differential equation are given by

\[ y_{n+1} = y_n + hf_{n+1}, \] (2)
\[ y_{n+1} = \frac{4}{3}y_n - \frac{1}{3}y_{n-1} + \frac{2}{3}hf_{n+1}, \] (3)
\[ y_{n+1} = \frac{18}{11}y_n - \frac{9}{11}y_{n-1} + \frac{2}{11}y_{n-2} + \frac{6}{11}hf_{n+1}. \] (4)

where we have used the conventions \( y_n = y(t_n) \) and \( f_{n+1} = f(t_{n+1}, y_{n+1}) \).

Using a forward difference approximation to the derivative we find that

\[ f(t_{n+1}, y_{n+1}) = \frac{y_{n+1} - y_n}{h}, \] (5)

where \( y^* \) is the value we are trying to improve. The BDF method (3) becomes

\[ y_{n+1}^{*} = \frac{2}{3}y_{n+1} + \frac{2}{3}y_{n} - \frac{1}{3}y_{n-1}. \] (6)

Similarly, by approximating the derivative by a central difference approximation we obtain

\[ y_{n+1}^{*} = \frac{1}{3}y_{n+1}^{*} + \frac{4}{3}y_{n} - \frac{2}{3}y_{n-1}. \] (7)

As stated in their names, BDF methods are backward approximations of the first order derivative in a first order ordinary differential equation. In this instance, however, we are not applying the BDF method to an ODE but rather to actual discrete data points. Equations (2)–(4), being discrete, are applicable when considering actual data instead of an ODE as is usually the case. Since we do not have a function \( f \), as per (1), we take an approximation of the first order derivative as per (5). The BDF method approximates a differential equation whereas finite difference approximations could be said to be a means of approximating the gradient making it a convenient way of approximating \( f \). This means that a lower order approximation of a gradient, that is, the finite difference approximation of \( f \), is incorporated into a higher order approximation of what we can term our discrete ODE.

Thus (6) mixes first and second order approximations of the first order derivative. Equation (7) mixes a centred approximation at position \( x_n \) and a backward approximation at position \( x_{n+1} \) which is equivalent to a 0 order approximation. The reason why all the terms do not cancel out in formulae (6) and (7) and why this approach is still relevant is due to the fact that the formulae mix different order approximations. Our purpose is to improve upon already obtained predictions—or an already obtained solution. More precisely our predictions are our \( y \) function and it is exactly the \( y \) function as discrete points which we aim to improve upon in the same way predictor correctors are used to improve upon solutions obtained via other numerical methods such as Euler’s method. Hence we are in fact not solving the ODE again, only improving on already known data in a pointwise fashion.

It is important to note that we are not implementing the BDF method in a direct fashion. We are incorporating...
a lower order approximation into the method in order to obtain a means of improving on already generated data. In the computational implementation of this work, the BDF method given by (6) is iterated to convergence. The initial value \( y_{n+1}^{(0)} = y_{n+1} \) is obtained from either a linear or spline interpolation. The value of \( y_{n+1} \) can be obtained using a neural network as discussed in other literature [14–16].

3. Convergence Properties

In this section we investigate the convergence properties of the difference equations (6) and (7). We want to show that

\[
\left| y_{n+1}^{(i+1)} - y_{n+1}^{(j)} \right| \to 0, \tag{8}
\]

by considering the general solution to (6) which is

\[
y_{n+1}^{(j)} = 2y_n - y_{n-1} + \left( \frac{2}{3} \right)^j \left( y_{n+1}^{(0)} - 2y_n + y_{n-1} \right). \tag{9}
\]

By considering (9) as follows

\[
\left| y_{n+1}^{(i+1)} - y_{n+1}^{(j)} \right| = \left( \frac{1}{3} \right)^j \left( y_{n+1}^{(0)} - 2y_n + y_{n-1} \right), \tag{10}
\]

we find that as \( j \to \infty \) (8) holds. In a similar fashion it is relatively easy to show that (8) is satisfied for (7), indicating convergence. In fact, scheme (7) converges faster than (6) since the coefficient of \( y_{n+1}^{(j)} \) is smaller.

When either (6) or (7) is iterated to convergence we have

\[
y_{n+1}^{(i+1)} = y_{n+1}^{*} \pm \epsilon = y_{n+1}^{*} \pm \epsilon, \text{ where } \epsilon \ll 1 \text{ is the tolerance.}
\]

For both (6) and (7) we find that at convergence

\[
y_{n+1}^{*} = 2y_n - y_{n-1} \pm \epsilon. \tag{11}
\]

It is in fact the right hand side of (11) which indicates the “correction” made to the original prediction. By implementing a lower order approximation within a higher order approximation as we have done in the previous section we have been able to obtain this necessary difference equation which is the manner in which the predictions are improved upon.

The results and concluding remarks are presented in the next section.

4. Results and Concluding Remarks

As a means of evaluating the effectiveness of the BDF method as implemented above to improve upon the prediction of the direction of data we compare the accuracy of the originally fitted data and the corrected data. Direction success rate can be seen as a simple bimodal result. If we let \( y_{n+1} \) be the true value then

\[
y_{n+1} = \begin{cases} \text{positive} \Rightarrow \text{success,} & \text{positive} \Rightarrow \text{success,} \\ \text{negative} \Rightarrow \text{failure.} & \text{negative} \Rightarrow \text{failure.} \end{cases}
\]

Choosing a sign based criterion is a standard method in financial analysis. The criteria are appropriate within the context of stock market modelling given the growing interest in developing tools which can give a sense of the direction in which the market may move or when possible turning points or shocks in returns will occur. This information is critical irrespective of the actual numerical value assigned to the movement given that the numerical values of the data used and analysed within this context are not necessarily useful in identifying characteristics which can be used to predict future movements. Hence, if the return on a stock decreases from 0.15 to 0.14, that is, by 0.01, for example, a prediction of −1 is of more value than a prediction of +0.01. This is due to the fact that the former prediction indicates the market trend whereas the latter only indicates a numerical value which within the context of market modelling loses value since it indicates a rise in the return at the next time step which is false. Thus the numerical value does not hold as much value in market trend trading as does the direction indicated by the sign of our prediction. In terms of our criterion, we have simply followed convention based upon methods currently employed by traders who trade according to market directions or trends.

In Table 1 we compare the percentage accuracy of predictions obtained through fitting a linear curve or spline through four points and the improved predictions found via (6). On average, this simple approach ensures that we get the direction correct 87% of the time in comparison to 48% for a linear curve and 86% when compared to 52% for a spline curve. These percentages have been calculated by simply counting the amount of times the sign of the actual point is generated, such as the three samples represented in Figure 1, and either the fitted or corrected point is the same and divided by the overall number of points. The number of points used—as indicated before—was 100. This is an arbitrary choice simply done for convenience.

Table 1 reflects the implementation of (6) obtained from the BDF method (3) and by using a forward difference approximation for the first derivative. If we instead incorporate a central difference approximation as per (7) we are able to improve a direction success rate of 50% for a linear curve to 80%. A comparison done against the spline fitting showed an improvement from 50% to 79%. We have also considered (4) with a forward and central difference approximation for the first derivative, respectively. The former choice showed an increase from approximately 48% to 80% for the linear comparison and 49% to 79% for the spline comparison. The latter case indicated an increase in the accuracy of the direction from 50% to 81% when the original predictions are obtained via a linear fitting and 49% to 81% when a spline is implemented.

In this paper we have shown how the implementation of the BDF method with a lower order approximation of the gradient of discrete data can improve the accuracy of predicting the direction of random data. The motivation for using a numerical ordinary differential equation solver comes from the fact that direction is just a gradient. We form a discrete ordinary differential equation from our vector of predictions. This discrete ordinary differential equation is solved with the implementation of a lower order approximation of the gradient with a BDF method. We find that the accuracy of our predictions improves from an accuracy of approximately 50% to an accuracy of approximately 87%.
Table 1: Table comparing percentage accuracy of predicting direction.

<table>
<thead>
<tr>
<th>Linear</th>
<th>Spline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitted Corrected</td>
<td>Fitted Corrected</td>
</tr>
<tr>
<td>51.0417% 80.0000%</td>
<td>56.2500% 88.4211%</td>
</tr>
<tr>
<td>52.0833% 92.6316%</td>
<td>50.0000% 86.3158%</td>
</tr>
<tr>
<td>52.0833% 90.5263%</td>
<td>52.0833% 86.3158%</td>
</tr>
<tr>
<td>45.8333% 87.3684%</td>
<td>47.9167% 88.4211%</td>
</tr>
<tr>
<td>55.2083% 92.6316%</td>
<td>41.6667% 85.2632%</td>
</tr>
<tr>
<td>46.8750% 85.2632%</td>
<td>53.1250% 91.5789%</td>
</tr>
<tr>
<td>48.9583% 84.2105%</td>
<td>48.9583% 85.2632%</td>
</tr>
<tr>
<td>42.7083% 85.2632%</td>
<td>59.3750% 86.3158%</td>
</tr>
<tr>
<td>35.4167% 87.3684%</td>
<td>44.7917% 76.8421%</td>
</tr>
<tr>
<td>48.9583% 88.4211%</td>
<td>52.0833% 85.2632%</td>
</tr>
</tbody>
</table>

An advantage of the approach taken in this paper is that the BDF scheme is a marching scheme. Irrespective of how big the data set is, the scheme will march through the data accordingly. The “speed” of the algorithm on very large data sets can be improved upon by using a computer with a faster CPU.

Acknowledgment

E. Momoniat and C. Harley acknowledge support from the National Research Foundation of South Africa.

References
