

On the use of BDF methods to improve the prediction of direction in market related data

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

Centre for Differential Equations, Continuum Mechanics and Applications, School of Computational and Applied Mathematics, University of the Witwatersrand, Johannesburg, Private Bag 3, Wits 2050, South Africa.

Abstract

The use of a BDF method as a tool to correct predictions made using curve fitting techniques is investigated. Random data having the same properties as the data we are modelling is generated. The data is assumed to have memory. Data in the period of the memory is curve fitted and then a prediction is made for the next time step. A vector of predictions is generated. The vector of predictions is converted into a discrete ordinary differential representing the gradient of the data. The BDF method is used to solve this discrete ordinary differential equation. The use of the BDF method improves the prediction of the direction by approximately 30%.

Key words: BDF method, random walk, predictions.

1 Introduction

In this brief note we show how a BDF method can be used as a corrector for predictions. The predictions aim to accurately reflect 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 there is an aspect of 'memory' in the data, i.e. information embedded in previous data points is still contained within the data points to be predicted. 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 percentage of accuracy in predicting the direction the data takes. BDF methods are backward differentiation formulae designed to solve initial value differential equations.

¹ Corresponding author. E-mail address: Ebrahim.Momoniat@wits.ac.za

They are constructed by satisfying the differential equation exactly at one point and then interpolating the previous points. A Lagrangian interpolation is typically used. The initial prediction of direction is made using linear/spline curve fitting. The novelty of the approach taken here is that we iterate the BDF formula to convergence.

Theoretical developments in mathematics of finance have centered around the random walk hypothesis [8,11] and the fact the market cannot be predicted when using this hypothesis. Various modifications to this hypothesis have been proposed with the development of the theory of Martingales [5,15]. The validity of the random walk hypothesis has been questioned in many studies. Most prominent among these is the book by Lo and MacKinlay [10].

While the mathematical theory used to develop the mathematical aspects of finance have not really focused on predicting returns there has been a strong interest in developing tools which can give a sense of the direction the market is going to take or when possible turning points will occur. The inclusion of ideas from the social sciences in financial mathematics has heralded the potential development of tools that can be used to aid the prediction of market trends. Among these ideas are aspects of behavioral science [6,9] and the notions of overreaction, underreaction and contrarian strategies [4,2,3,7,9]. Berman [1] has attempted one of the first studies to analyse the Global Real Estate Securities market using aspects of these contrarian ideas.

The paper is set out as follows: in Section 2 we develop and motivate the algorithm. Convergence properties of the algorithm are discussed in Section 3. Results and concluding remarks are presented in Section 4. We include an appendix with the algorithm in MATLAB at the end of the paper.

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 data in the interval zero-one with zero mean and standard deviation of one. We start out with an element that has value zero. 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`. This is a typical random walk. We continue stepping in this way a finite number of times. The last data point is put into a vector we use to simulate our times series of returns. We continue running the random walk until we have a vector of returns. The vector of returns is normalised by subtracting the mean and dividing by the variance. The return data generated in this way has zero mean

with standard deviation smaller than one. In Figure 1 we plot three simulations of return data obtained in this way. We have found that the data obtained in this way simulates the actual time series very well.

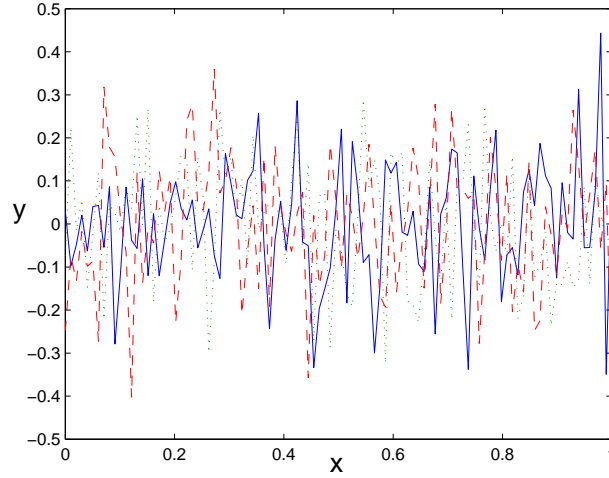


Fig. 1. *Plot of simulated data.*

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. We then curve fit through the initial four points in the vector representing actual data. We use this fitted curve to predict the value of the fifth point. We then use data values two to five to predict the sixth value. Continuing in this way we end up with a vector of actual values and a vector of predicted values - four data points shorter than the original. We then calculate the percentage of times the sign of the predicted data matched the sign of the actual data. From Table 1 we observe that curve fitting predicts direction correctly approximately 50% of the time.

To improve the accuracy of predicting direction we make use of the fact that the direction is just the gradient of the data. By creating a vector of gradients we have the numerical representation of an ordinary differential equation. We then use a BDF method to numerically solve the ordinary differential equation. BDF methods are appropriate because it depends on previous values. Some examples of BDF methods for solving the first-order ordinary differential equation

$$y' = f(x, y), \quad (2.1)$$

are given by

$$y_{n+1} = y_n + hf_{n+1}, \quad (2.2)$$

$$y_{n+1} = \frac{4}{3}y_n - \frac{1}{3}y_{n-1} + \frac{2}{3}hf_{n+1}, \quad (2.3)$$

$$y_{n+1} = \frac{18}{11}y_n - \frac{9}{11}y_{n-1} + \frac{2}{11}y_{n-2} + \frac{6}{11}hf_{n+1}, \quad (2.4)$$

where we have used the convention $y_n = y(x_n)$ and $f_{n+1} = f(x_{n+1}, y_{n+1})$.

Using a forward difference approximation to the derivative we find that

$$f(x_{n+1}, y_{n+1}) = \frac{y_{n+1}^* - y_n}{h} \quad (2.5)$$

where y^* is the value we are trying to improve. The BDF method (2.3) becomes

$$y_{n+1}^{*(j+1)} = \frac{2}{3}y_{n+1}^{*(j)} + \frac{2}{3}y_n - \frac{1}{3}y_{n-1}. \quad (2.6)$$

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

$$y_{n+1}^{*(j+1)} = \frac{1}{3}y_{n+1}^{*(j)} + \frac{4}{3}y_n - \frac{2}{3}y_{n-1}. \quad (2.7)$$

In the example code we implement the BDF method (2.6) which we iterate 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, [12–14].

3 Convergence properties

In order to investigate the convergence properties of the difference equations (2.6) and (2.7) we consider the difference equation

$$Y_{j+1} = \alpha Y_j + \beta, \quad (3.8)$$

where $Y_j = y_{n+1}^{*(j)}$ and β a constant. The difference equation (3.8) has the general solution

$$Y_j = \alpha^j Y_0 + \beta \sum_{i=0}^{j-1} \alpha^i. \quad (3.9)$$

We want to show that

$$|Y_{j+1} - Y_j| \rightarrow 0 \quad (3.10)$$

as j increases. From (3.9) we find that

$$|Y_{j+1} - Y_j| = |\alpha^j| |Y_0(\alpha - 1) + \beta|. \quad (3.11)$$

For the iterative scheme (2.6) we have $\alpha = 2/3$ and $\beta = (2/3)y_n - (1/3)y_{n-1}$ while for (2.7) we have $\alpha = 1/3$ and $\beta = (4/3)y_n - (2/3)y_{n-1}$. In both cases $\alpha < 1$ and hence $\alpha^j \rightarrow 0$ for large j . Therefore, it is relatively easy to show that (3.10) is satisfied and our iterative schemes (2.6) and (2.7) converge. In fact, the scheme (2.7) converges faster than (2.6) since the coefficient of $y_{n+1}^{*(j)}$ is smaller.

When either (2.6) or (2.7) is iterated to convergence we have $y_{n+1}^{*(j+1)} = y_{n+1}^{*(j)} \pm \epsilon = y_{n+1}^* \pm \epsilon$ where $\epsilon \ll 1$ is the tolerance. For both (2.6) and (2.7) we find that at convergence

$$y_{n+1}^* = 2y_n - y_{n-1} \mp \epsilon. \quad (3.12)$$

Direction success rate can be seen as a simple bimodal result. If we let y_{n+1}^T be the true value then

$$y_{n+1}^T y_{n+1}^* = \begin{cases} \text{positive} \implies \text{success,} \\ \text{negative} \implies \text{failure.} \end{cases} \quad (3.13)$$

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

4 Results and Concluding remarks

On average, this simple approach ensures we get direction correct 80% of the time. We note from the table that by using the BDF as a corrector we have improved the direction from an average of 50% to an average of 80%.

| Linear | | Spline | |
|---------------|-----------|---------------|-----------|
| Fitted | Corrected | Fitted | Corrected |
| 51.0417 % | 80.0000 % | 56.2500 % | 88.4211 % |
| 52.0833 % | 92.6316 % | 50.0000 % | 86.3158 % |
| 52.0833 % | 90.5263 % | 52.0833 % | 86.3158 % |
| 45.8333 % | 87.3684 % | 47.9167 % | 88.4211 % |
| 55.2083 % | 92.6316 % | 41.6667 % | 85.2632 % |
| 46.8750 % | 85.2632 % | 53.1250 % | 91.5789 % |
| 48.9583 % | 84.2105 % | 48.9583 % | 85.2632 % |
| 42.7083 % | 85.2632 % | 59.3750 % | 86.3158 % |
| 35.4167 % | 87.3684 % | 44.7917 % | 76.8421 % |
| 48.9583 % | 88.4211 % | 52.0833 % | 85.2632 % |

Table 1

Table comparing percentage accuracy of predicting direction.

In this paper we have shown how a BDF method 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 using a BDF method. We find that the accuracy of our predictions improves from an accuracy of approximately 50% to an accuracy of approximately 80%.

For the return times series data we are modelling we compared a range of periods over which we believed the data had memory. The period that minimized the maximum absolute error between prediction and actual was then chosen. The predictions were corrected using a BDF method. We did not use curve fitting for our predictions, but instead developed our own approach based on neural networks. By successfully being able to predict the direction of the aspect of the South African market we are interested in we are able to develop models that are useful for the hedge fund industry.

Acknowledgments

EM acknowledges support from the National Research Foundation, South Africa, under grant number 2053745.

References

- [1] M. Berman, Cum Dividend Irrational Pricing Behaviour in US REITs, PhD thesis, Rushmore University, (2007).
- [2] L. K. C. Chan, N. Jegadeesh, J. Lakonishok, Momentum Strategies, *The Journal of Finance*, **51(5)** (1996) 1681–1713.
- [3] K. Daniel, D. Hirshleifer, A. Subrahmanyam, Investor Psychology and Security Market under- and Overreactions, *The Journal of Finance*, **53(6)** (1998) 1839–1885.
- [4] W. F. M. De Bondt, R. Thaler, Does the Stock Market Overreact? *The Journal of Finance* **40(3)**, Papers and Proceedings of the Forty-Third Annual Meeting American Finance Association, Dallas, Texas, December 28-30, 1984, (1985) 793–805.
- [5] W. Feller, An Introduction to Probability Theory and its Applications, Vol 1, 1968.
- [6] H. Fromlet, Behavioral Finance-Theory and Practical Application, *Business Economics* (2001) 63.
- [7] H. Hong, J. C. Stein, A Unified Theory of Underreaction, Momentum Trading, and Overreaction in Asset Markets, *The Journal of Finance* **54(6)** (1999) 2143–2184.

- [8] S. M. Keane, *Stock Market Efficiency*, Oxford, Philip Allan Limited, 1983.
- [9] J. Lakonishok, A. Shleifer; R. W. Vishny, Contrarian Investment, Extrapolation, and Risk, *The Journal of Finance* **49(5)** (1994) 1541–1578.
- [10] A. W. Lo and A. C. Mackinlay, *A Non-Random Walk Down Wall Street*. 5th ed. Princeton: Princeton University Press, 2002, 4–47.
- [11] B. G. Malkiel, *A Random Walk Down Wall Street*. 6th ed. New York: W.W. Norton & Company, Inc., 1973.
- [12] K. Chakraborty, K. Mehrotra, C. K. Mohan and S. Ranka, Forecasting the behavior of multivariate time series using neural networks, Neural Networks archive, *Elsevier Science Ltd*, **5(6)** (1992), 961–970.
- [13] G. Zhanga, M. Y. Hu, B. E. Patuwo and D. C. Indrob, Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis, *Elsevier Science*, **116(1)**, (1999), 16–32.
- [14] Z. Tang and P. A. Fishwick, Feed-forward neural nets as models for time series forecasting, *ORSA Journal of Computing*, **5**, (1993), 374–385.
- [15] D. Williams, *Probability with Martingales*, Cambridge University Press, 1991.

Appendix

The code below is used to generate our random data where n is the number of points generated in the random walk and m is the number of points in our vector of returns. The function `randn` produces pseudo-random number from a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$.

```

for j=1:m;
x=zeros(1,n);
x(1)=0;

for i=1:n-1;
z=randn;
if z<.5
x(i+1)=x(i)+z;
else
x(i+1)=x(i)-z;
end
end
y(j)=x(n);
end

```

The end result of this part of the code is that we have data that looks like the return data we are modelling. We normalize the data by subtracting the mean and dividing by the variance.

```
y=(y-mean(y))./var(y);
```

In Figure 1 we plot 3 runs of the code above. In the next part of the code we use a 'linear' interpolation to create a vector of prediction. Linear in the code below can be changed to 'spline' for a comparison with a cubic spline. The variable `noofpts` is determined by the memory one assumes in the data. We choose `noofpts=4` for the purpose of demonstrating the code.

```
noofpts=4;
p=m-noofpts;
for i=1:p;
newy(i)=ppval(interp1(x(i:i+noofpts-1),y(i:i+noofpts-1),...
'linear','pp'),x(i+noofpts)));
end
```

We resize the original data to be the same length as the interpolated data.

```
oldy=y(noofpts+1:n);
```

We then calculate the number of times the 'linear' or 'spline' has correctly predicted the direction of the data.

```
count=0;
for i=1:p;
if newy(i)*oldy(i)>0
count=count+1;
end
end
```

In the next part of the code the BDF method is used to improve on the predictions. The BDF method is iterated to convergence.

```
for j=2:p;
newnewy(1,j-1)=newy(j);
oldnewy=10;
while abs(newnewy(1,j-1)-oldnewy)>10^(-4);
oldnewy=newnewy(1,j-1);
newnewy(1,j-1)=(4/3)*oldy(j)-(1/3)*oldy(j-1)+(2/3)*(newnewy(1,j-1)-oldy(j));
```

```
end
newyval(j-1)=newnewy(1,j-1);
end
```

We resize the vector of prediction to be the same size as the corrected predictions.

```
oldy=oldy(2:p);
```

Once again we calculate the number of times the BDF has correctly predicted the direction of the data.

```
ncount=0;
for i=1:p-1;
if newyval(i)*oldy(i)>0
ncount=ncount+1;
end
end
```

The information is then converted into percentages.

```
disp([count/p*100,ncount/(p-1)*100])
```