Neural Networks 156 (2022) 95–107

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Time series (re)sampling using Generative Adversarial Networks[☆]

Christian M. Dahl[a,*], Emil N. Sørensen[b]

[a] *Department of Business and Economics, University of Southern Denmark, Denmark*
[b] *School of Economics, University of Bristol, United Kingdom*

A R T I C L E I N F O

Article history:
Received 23 March 2021
Received in revised form 7 September 2022
Accepted 8 September 2022
Available online 23 September 2022

Keywords:
Generative adversarial nets
Dependent processes
Bootstrapping

A B S T R A C T

We propose a novel bootstrap procedure for time series data based on Generative Adversarial networks (GANs). We show that the dynamics of common stationary time series processes can be learned by GANs and demonstrate that GANs trained on a single sample path can be used to generate additional samples from the process. We find that temporal convolutional neural networks provide a suitable design for the generator and discriminator, and that convincing samples can be generated on the basis of a vector drawn from a normal distribution with zero mean and an identity variance–covariance matrix. We demonstrate the finite sample properties of GAN sampling and the suggested bootstrap using simulations where we compare the performance to circular block bootstrapping in the case of resampling an AR(1) time series processes. We find that resampling using the GAN can outperform circular block bootstrapping in terms of empirical coverage. Finally, we provide an empirical application to the Sharpe ratio.

1. Introduction

Generative Adversarial Nets (GANs) were introduced by Goodfellow et al. (2014). Based on an initial training sample, GANs learn to generate additional data that "looks" similar. GANs and their applications are intensely researched in the deep learning literature (Arjovsky, Chintala, & Bottou, 2017; Gulrajani, Ahmed, Arjovsky, Dumoulin, & Courville, 2017; Salimans et al., 2016). We propose a new bootstrap for time series processes based on GAN. While GANs have been used to generate time series data, we show that by using a GAN, we can bootstrap from an existing sample path of a time series process. To our knowledge, GANs have not been applied to this problem before. We show that, in terms of empirical coverage, our GAN-based procedure yields finite sample improvements over traditional bootstrapping methods that do not rely on GANs.

The intuitive appeal of time series GANs is related to the Wold Representation Theorem (WRT) for time series proposed by Wold (1939). By the WRT, any stationary time series process can be represented as an infinite moving average process. In principle, this implies that any stationary time series can be constructed based on a sequence of white noise random variables. GANs use noise to generate new data that mimic an existing sample. Hence, given samples of white noise and an observed sample path from a time series process, carefully designed GANs should be able to uncover and replicate the properties of any stationary time series process, which can then be used for bootstrapping.

GANs have recently received significant attention in the context of time series analysis.[1] Hyland, Esteban, and Rätsch (2017) applied GANs, based on recurrent neural networks (RNN), to what they call "medical" time series. Yoon, Jarrett, and van der Schaar (2019) proposed TGAN, a generative time series model, that is also based on an RNN architecture. Wiese, Knobloch, Korn, and Kretschmer (2020) described a GAN for financial time series and show that it can reproduce the stylised facts of such series. Their GAN uses temporal convolutional networks which are related to the architecture we suggest in this work. Smith and Smith (2020) proposed TSGAN to generate time series in a few-shot setting. They base their model on generation of spectrograms which are translated into realisations of the time series. Ni et al. (2021)

☆ Acknowledgements: The authors gratefully acknowledge support from the Google Tensorflow Research Cloud (TFRC). Code for this paper is available on request. We thank Giovanni Mellace and Peter Sandholt Jensen for useful comments. We also thank Haixia Bi, Torty Sivill, and Hao Song for comments on an earlier draft of this paper. We are grateful to the editor, the associate editor, and four anonymous referees for useful comments and suggestions that significantly improved this paper. We are also grateful to Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer for kindly and quickly sharing their code for QuantGAN. Any errors are our own.

* Corresponding author.
E-mail addresses: cmd@sam.sdu.dk (C.M. Dahl), e.sorensen@bristol.ac.uk (E.N. Sørensen).

[1] Traditionally, and less relevant to our work, GANs have been applied in image processing and computer vision, natural language processing and in generation of music, speech and audio, see, e.g., Gui, Sun, Wen, Tao, and Ye (2020). Somewhat more related to our work, there are also applications of GANs to the estimation of structural models and treatment effects (Athey, Imbens, Metzger, & Munro, 2021; Kaji, Manresa, & Pouliot, 2018).

https://doi.org/10.1016/j.neunet.2022.09.010

proposed a novel generator architecture, called the conditional AR-FNN, that is designed to capture the temporal dependence of time series. They also propose a new metric, (conditional) Sig-W1, that captures the (conditional) joint law of time series models, and use this metric as a discriminator. Sun, Deng, Chen, and Parkes (2020) develop the Decision-Aware Time series GAN (DAT-CGAN). They develop a generator that can assist, for example, financial managers, in decisions about their portfolios. Some managers might be more risk adverse than others and care more about higher order moments in the underlying financial assets than others. The DAT-CGAN generator can be tailored to support such individual and heterogeneous preferences. For a more comprehensive survey on GAN models and time series, see Brophy, Wang, She, and Ward (2021).

Unlike the existing work we outlined above, our contribution focuses on the use of GANs in a bootstrap-like method for time series. The potential usefulness of GANs for such time series bootstraps is briefly mentioned in recent work by Haas and Richter (2020). Bootstrapping dependent data, such as samples from a time series process, has received long-standing attention in the time series literature, see, e.g., the overview by Lahiri (1999) and the newer contributions of Paparoditis and Politis (2001) and Shao (2010). We suggest that GANs provide a novel approach to such bootstrapping which we call the generative bootstrap (GB). The theory of GANs, although not GANs applied to bootstrapping, is still being developed and is an active area of research, see Biau, Cadre, Sangnier, and Tanielian (2018), Biau, Sangnier, and Tanielian (2021) and Haas and Richter (2020). As the theory of GANs and standard GAN (independent and identical distributed (iid)) bootstrapping is not well-developed, we do not aim to extend the theory to time series GAN bootstraps. Instead we contribute by analysing the finite sample properties of generative bootstrapping using simulations where we compare the performance against Circular Block Bootstrapping (CBB) (Politis & Romano, 1992) for the AR(1) process.

The contributions of our paper are threefold and can be summarised as follows: First, using simulated data, we recover the parameters of the true data generating process and thereby show that the GAN has learned at least a minimum of characteristics of the process, which is in line with the existing work on time series GANs. Second, and as our main contribution, we find, in simulations, that our novel GB procedure, which uses the GAN for re-sampling, can outperform the classical CBB for dependent data on the empirical coverage of percentile confidence intervals. Finally, we apply our novel GB to bootstrap the Sharpe ratio, and we adapt an existing time series GAN, called QuantGAN (Wiese et al., 2020), to bootstrapping, and use this for comparison. We also compare against the canonical time series CBB (Politis & Romano, 1992).

In Section 2, we briefly review two common GANs and discuss how they can be trained to generate samples from a time series based on an initial sample path. Section 3 discusses an algorithm for using the GAN to bootstrap dependent data. Section 4 provides simulation evidence of the quality of the trained GAN and the performance when it is used for bootstrapping. Section 5 gives an empirical illustration in the context of the Sharpe ratio, and we compare against an adapted QuantGAN-based bootstrap procedure and the CBB. Section 6 concludes.

2. Generative adversarial nets

The concept of GANs can be introduced intuitively as follows. Assume that a real sample of data is drawn from the unknown distribution F_X, and assume that we have another arbitrary, but known, distribution F_Z. The generator G is a function that transforms a sample from F_Z into a sample that looks like it is drawn from the real data distribution F_X. The discriminator D is a function that tries to determine if a given sample is drawn from the real data distribution F_X or not. The two models are set to play a game against each other. The generator tries to fool the discriminator by generating fake samples that look as real as possible, and the discriminator tries to detect the generator's forgery by determining if it got a real or fake sample.

Let G and D be specified up to the finite dimensional parameters θ_G and θ_D, respectively. Also, let x_{real} denote some generic real sample from distribution F_X, and $x_{fake} = G(z;\theta_G), z \sim F_Z$ a generated sample. Goodfellow et al. (2014) suggest solving the minimax problem

$$\min_{\theta_G}\max_{\theta_D} \ \mathbb{E}_{F_X}\log D(x;\theta_D) + \mathbb{E}_{F_Z}\log(1 - D(G(z;\theta_G);\theta_D)), \tag{1}$$

where the generator solves the minimisation problem, and the discriminator solves the maximisation problem. Biau et al. (2018) and Goodfellow et al. (2014) argue, under a set of assumptions, that the optimal discriminator in the minimax formulation in Eq. (1) is related to the Jensen–Shannon divergence between the distributions of the real and generated data. Arjovsky et al. (2017) argue that an alternative distance measure between the distributions is the order-1 Wasserstein (Earth-Mover) distance which results in the Wasserstein GAN. For the Wasserstein GAN, Arjovsky et al. (2017) suggest solving the minimax problem

$$\min_{\theta_G}\max_{\theta_D} \ \mathbb{E}_{F_X}D(x;\theta_D) - \mathbb{E}_{F_Z}D(G(z;\theta_G);\theta_D) \tag{2}$$

subject to $D(\,\cdot\,;\theta_D)$ being Lipschitz with Lipschitz constant 1. Gulrajani et al. (2017) argue that, since a function is Lipschitz-1 if and only if the norm of the gradient is 1 or less everywhere, the Lipschitz condition could in practice be imposed as a soft constraint using a gradient penalty on D.

To state the loss functions, let $\{(z_i, x_{i,real})\}_{i=1}^{n_b}$ constitute a batch of data where z_i is noise sampled from F_Z and $x_{i,real}$ is a real sample. During training we minimise the batch discriminator loss

$$\begin{aligned} L_D^{(b)} &= \frac{1}{n_b}\sum_{i=1}^{n_b} D(x_{i,fake};\theta_D) - D(x_{i,real};\theta_D) \\ &\quad + \lambda\frac{1}{n_b}\sum_{i=1}^{n_b}\left(\|\nabla_{\tilde{x}}D(\tilde{x}_i;\theta_D)\|_2 - 1\right)^2, \\ \tilde{x}_i &= a x_{i,real} + (1-a)x_{i,fake} \end{aligned} \tag{D1}$$

where $x_{i,fake} = G(z_i;\theta_G)$, and the last term is the gradient penalty on D (Gulrajani et al., 2017). The gradient penalty has weight λ, $\|\cdot\|_2$ denotes the l_2 norm, and $\tilde{x} = a x_{real} + (1-a)x_{fake}$ is a convex combination of x_{real} and x_{fake} with uniform random weight $a \sim \mathrm{U}(0,1)$, as described in Gulrajani et al. (2017). As per usual, the batch gradients $\nabla_{\theta_D}L_D^{(b)}$ serve as unbiased estimates of $\nabla_{\theta_D}L_D$ which allow us to do stochastic gradient descent on the parameters (θ_D, θ_G). Similarly, for the generator we minimise the batch generator loss

$$L_G^{(b)} = \frac{1}{n_b}\sum_{i=1}^{n_b} -D(G(z;\theta_G);\theta_D). \tag{G1}$$

By alternating between the objectives (D1) and (G1), we can learn the parameters of G and D. A complete training algorithm is available in Gulrajani et al. (2017).

2.1. Applying GANs to time series

The GAN formulation above does not necessarily impose how we should parameterise the discriminator D and generator G. However, in practice, they are commonly learned using neural networks with exact parameterisations depending on the application. In this work, we model time series processes, which imposes some natural restrictions on the architecture.

A time series is a sequence of observations ordered by time, say, $Y = \{Y_t : t \in \mathcal{T}\}$, where t indexes time and $\mathcal{T}$ denotes the index set. A time series has the defining property that information flow is unidirectional, so the state of the process at time t, Y_t, depends only on the past information $(Y_{t-1}, Y_{t-2}, \dots)$ and not the future $(Y_{t+1}, Y_{t+2}, \dots)$. This constraint is useful when we choose the parameterisations of G and D.

We recognise that there are many potential choices of GAN architecture for modelling time series data.[2] Our architecture only serves to illustrate the procedure we introduce in Section 3. We base our generator and discriminator on dilated temporal convolutional (TC) networks as used by Oord et al. (2016) in the context of audio time series. We will refer to this as the TC-architecture. The temporal convolutional networks enforce the unidirectional flow of information. They were applied to time series forecasting by, e.g., Borovykh, Bohte, and Oosterlee (2017) and Sen, Yu, and Dhillon (2019). In particular, Borovykh et al. (2017) showed that TC networks outperform RNNs in several forecasting problems and are easier to train even for long-range dependence. Recently, Wiese et al. (2020) also suggested temporal convolutional networks based on Oord et al. (2016) for financial time series modelling with GANs. The dilation of the temporal convolutional layers increases the receptive field, in context of time series this is the number of lags that the model can accommodate at once, while limiting the number of parameters (Borovykh et al., 2017; Oord et al., 2016).

In practice, we implement the TC network as a network of conventional one-dimensional convolutional layers with appropriate zero-padding applied to the input, as described by Oord et al. (2016). If we stack d TC layers with kernel size 2, where the dilation for layer i is 2^i, then the total receptive field size at the final layer will be $p = \sum_{i=1}^{d} 2^i = 2^{d+1} - 1$ (Yu & Koltun, 2016). For illustration, assume that our generator G consists of d TC layers. To generate a time series of length b, we slide the TC layers over a sequence of $(b + p)$ independent and identically distributed (iid) noise terms $(z_1, z_2, \dots, z_{p+b})$, $z_t \sim F_Z$ where F_Z is some known distribution and p is the receptive field size. During the GAN training, the parameters in the TC layers learn to transform this sequence of iid noise into observations from the time series. This is illustrated in Fig. 1a for a generator with two TC layers. Notice that a TC network of fixed size can generate arbitrarily long sequences, and that the lengths of the generated sequences are controlled by the number of noise terms supplied to the generator. The discriminator D considers sequences of observations from the generated or real time series $(y_1, y_2, \dots, y_T)$ and learns the parameters in the TC layers to distinguish between real and generated samples. The training setup is illustrated in Fig. 1b. The intuitive appeal of the TC architecture is related to the Wold Representation Theorem (WRT) for time series proposed by Wold (1939). By the WRT, any stationary time series process can be represented as an infinite moving average process. In principle, this implies that any stationary time series can be constructed based on a sequence of white noise random variables. GANs use noise to generate new data that mimic an existing sample. Hence, given samples of white noise and an observed sample path from a time series process, carefully designed GANs should be able to uncover and replicate the properties of any stationary time series process.

[2] Related work in this area are Brophy et al. (2021), Hyland et al. (2017), Koshiyama, Firoozye, and Treleaven (2021), Ni et al. (2021), Smith and Smith (2020), Sun et al. (2020), Wiese et al. (2020) and Yoon et al. (2019).

3. Generative bootstrap

We propose the GAN with temporal convolutional networks as a method to resample from a time series process. We use this in a bootstrap-like procedure to investigate the properties of estimators applied to time series data. This procedure is called the Generative Bootstrap (GB) and it is the main contribution of this paper. The GB is composed of two stages: (1) the GAN is trained on an initial sample from the true DGP using blocking – the *training stage*. (2) samples are generated from the generator of the trained GAN – the *sampling stage*.

The initial sample from the true DGP is re-sampled using a moving block scheme similar to the Moving Block Bootstrap (MBB) (Kunsch, 1989; Liu, Singh, et al., 1992) and n_b of such blocks constitute a batch of data that are used to perform one iteration over the batch losses in Eqs. (D1)–(G1). Many iterations are performed until the GAN losses stabilise. For sampling, the discriminator is discarded and we feed noise into the generator from an arbitrary distribution F_Z. The generator output is used as a sample to calculate one set of bootstrap statistics. This procedure is repeated for an arbitrary number of samples and the collection of statistics is used to form the GB estimates.

We note that this differs from conventional block bootstrapping in two respects. First, a conventional MBB would sample blocks with replacement from the initial sample and stack these into a sample path matching the length of the initial sample. This stacked sample is then used to calculate bootstrap statistics. In the GB, these blocks are not stacked, but fed as individual samples to train the GAN. Second, as we will detail later, under the TC-architecture, sampling the GAN does not have to use stacking and, once trained, the GAN can provide a sample of any size; even one that differs from the initial block size used for training. The size of the generated sample is determined by the noise terms supplied to the generator.

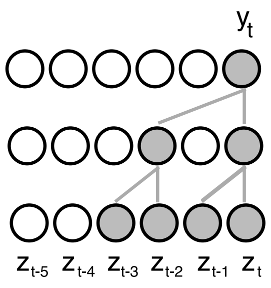
We proceed to discuss the training and sampling stages below.

Training stage. The training stage trains the GAN using moving blocks re-sampled from the initial sample. Let $y_i^* = (y_{1,i}^*, y_{2,i}^*, \dots, y_{T,i}^*)$ be the initial sample from the true DGP. We perform a blocking procedure identical to the moving block bootstrap. Define each of the $(T - b_1 - 1)$ overlapping blocks by $B_j^* = (y_{1j} = y_{1+j,i}^*, \dots, y_{2j} = y_{1+j+b_1,i}^*)$ where the block size is $b_1 < T$. A batch of training data is given by randomly sampling n_b blocks from $\{B_1^*, B_2^*, \dots, B_{T-b_1-1}^*\}$ without replacement. Denote this batch of true blocks by $\mathcal{Y}^*$.

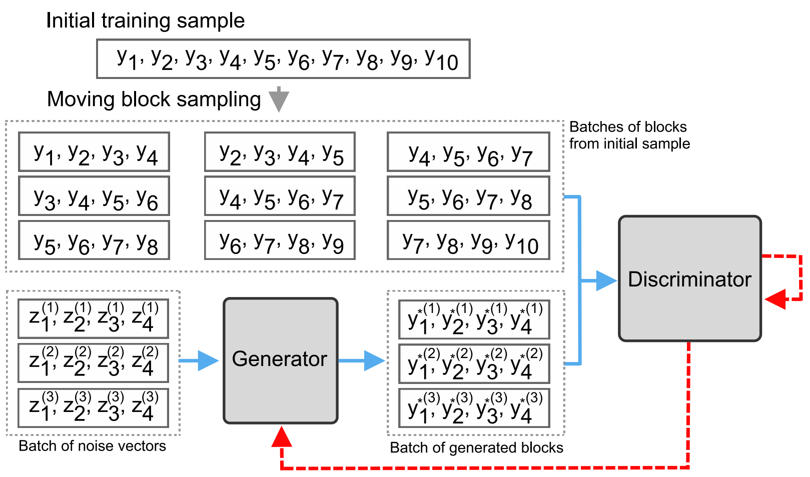
Next, we sample the generator noise from a multivariate standard normal distribution with an identity variance–covariance matrix, but the distribution F_Z could be selected arbitrarily. To generate a sample path of length b_1, we need $(b_1 + p)$ noise terms where p is the receptive field of the TC network. The noise terms are used to generate a block sample $B_j = (G(z_1, \dots, z_p; \theta_G), G(z_{1+1}, \dots, z_{p+1}; \theta_G), \dots, G(z_{b_1}, \dots, z_{b_1+p}; \theta_G))$, and n_b of such blocks are generated to form a batch of fake blocks. Denote this fake batch by $\mathcal{Y}$. The true and fake samples are fed to the discriminator and it tries to distinguish between them. This procedure of sampling true and fake blocks and feeding them to the discriminator constitutes the training stage. In practice, we iterate over Eqs. (D1)–(G1) until the losses stabilise. Eq. (D1) requires both a fake and true batch per iteration, while Eq. (G1) needs only a fake batch.

Sampling stage. Let $G(\cdot\,; \hat{\theta}_G)$ be the learned generator from the training stage. Once trained, the generator should produce samples mimicking the true DGP that generated the initial sample y_i^*. To generate a sample of length b_2, we first sample a sequence of noise vectors from F_Z

$$z_i = (z_{1,i}, z_{2,i}, \dots, z_{b_2+p,i}), \quad z_{t,i} \sim F_Z$$



(a) Temporal convolutional network


Initial training sample
y1, y2, y3, y4, y5, y6, y7, y8, y9, y10
Moving block sampling
y1, y2, y3, y4
y2, y3, y4, y5
y4, y5, y6, y7
y3, y4, y5, y6
y4, y5, y6, y7
y5, y6, y7, y8
y5, y6, y7, y8
y6, y7, y8, y9
y7, y8, y9, y10
Batches of blocks from initial sample
Discriminator
z1(1), z2(1), z3(1), z4(1)
z1(2), z2(2), z3(2), z4(2)
z1(3), z2(3), z3(3), z4(3)
Batch of noise vectors
Generator
y1*(1), y2*(1), y3*(1), y4*(1)
y1*(2), y2*(2), y3*(2), y4*(2)
y1*(3), y2*(3), y3*(3), y4*(3)
Batch of generated blocks

(b) GAN training

Fig. 1. Panel (a): An illustration of the temporal convolutional network. The output y_t at time t is a function of the present and past noise terms $(z_t, z_{t-1}, z_{t-2}, z_{t-3})$. We generate the output samples as we slide across the noise terms. Panel (b): The training stage of the GAN where it trains on blocks from an initial sample path of a time series process. Blue lines indicate the flow of training data. Red lines indicate back propagation. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

where p is the receptive field size. As in the training stage, F_Z is a multivariate standard normal distribution with an identity variance–covariance matrix. Any distribution could be used, the important point is that the training and sampling stages use the *same* distribution for F_Z. Next, we obtain a generated sample path y_i by passing the noise vectors through the learned generator,

$$y_i = (G(z_{1,i}, \dots, z_{p,i}; \hat{\theta}_G), G(z_{1+1,i}, \dots, z_{p+1,i}; \hat{\theta}_G), \dots,$$
$$G(z_{b,i}, \dots, z_{b_2+p,i}; \hat{\theta}_G)).$$

A single sequence of noise vectors $z_i = (z_{1,i}, \dots, z_{b_2+p,i})$ generates one sample path y_i of length b_2. We can repeat this process to obtain an arbitrary number of sample paths. Under the proposed TC-architecture, the generator can sample a block of any length from the underlying process. Hence, we are not restricted to the block size on which the model was trained, i.e. it is perfectly acceptable if $b_1 \neq b_2$. This does not necessarily hold for all choices of architecture, e.g., a fully-connected network would not have this property. This is a very attractive feature of the TC and GAN approach as it alleviates the need to stack individual blocks in a way that might break the dependence structure of the time series. We can simply choose the sampling block size to be equal to the size of the initial sample path, so $b_2 = T$. When $b_2 < T$ we refer to it as *blocked sampling*, while $b_2 = T$ is called *complete sampling*.

Bootstrap statistics. Let $G(\cdot; \hat{\theta}_G)$ be the learned generator from the training stage that has been trained on a single initial sample y_i^* from the true DGP. We now discuss how to calculate bootstrap statistics on the GAN samples. Assume that we are interested in parameter ϕ which has a suitable estimator $\hat{\phi}$. We use the sampling procedure from the previous section to obtain m samples from the learned generator $G(\cdot; \hat{\theta}_G)$, denote these samples by $(y_1, y_2, \dots, y_m)$ where $y_i = (y_{1,i}, \dots, y_{T,i}), i = 1, \dots, m$. Each y_i is considered a realisation of the DGP that produced the initial training sample for the GAN. We calculate the bootstrap statistics $\hat{\phi}_i \equiv \hat{\phi}(y_i), \quad i = 1, \dots, m$ resulting in m estimates $(\hat{\phi}_1, \hat{\phi}_2, \dots, \hat{\phi}_m)$. Similar to conventional bootstrapping (Efron, 1981), the GB variance estimate of $\hat{\phi}$ is

$$\hat{\sigma}_{GB,\hat{\phi}} = \frac{1}{m} \sum_i (\hat{\phi}_i - \hat{\phi}_{GB})^2. \tag{3}$$

The $(1-\alpha)$ GB confidence interval (CI) for ϕ is the $(1-\alpha)$ percentile CI (Efron, 1981) constructed using the empirical quantiles[3] $(\alpha/2, 1-\alpha/2)$ of $(\hat{\phi}_i)_i$,

$$\hat{I}_{\phi,1-\alpha} = \left[\hat{\phi}_{(\alpha/2)}, \ \hat{\phi}_{(1-\alpha/2)}\right]. \tag{4}$$

4. Simulations

In this section, we illustrate the performance of the GB by simulations and by making comparisons to the established CBB approach for bootstrapping dependent processes. While the simulations focus on the GAN architecture suggested in Section 2, we stress that the GB can use any suitable GAN.[4]

The simulation design is as follows. For simplicity, the true DGP is a zero mean and stable AR(1) process with normal distributed innovations:

$$y_t = \phi\, y_{t-1} + \epsilon_t, \quad \epsilon_t \sim \mathrm{N}(0, 1). \tag{5}$$

For each replication, a sample path of length $T = 1000$ is generated from Eq. (5). This sample is used to train the GAN with a training block size of $b_1 = 150$ and batch size $n_b = 64$. Once training is complete, we sample 10,000 sample paths from the GAN. These samples are used for two purposes:

First, in Section 4.1, we compare the samples generated by the proposed GAN to the known properties of the DGP under complete sampling with $b_2 = 1000$. We use the generated samples to estimate the autocorrelation (ACF) and partial autocorrelation (PACF) functions over 1000 replications. We consider three choices of the autoregressive parameter $\phi = 0.5, 0.8, 0.9$.

Second, in Section 4.2, we compare GB and the CBB for confidence interval estimation, i.e., empirical coverage, of the least-squares estimator $\hat{\phi}_{LS}$ of ϕ. We consider seven values of the

[3] The bias-correction techniques in Efron (1987) could potentially improve this confidence interval.

[4] In the Appendix, to illustrate a different choice of GAN for the GB, we use the recently suggested QuantGAN (Wiese et al., 2020), which is also based on temporal convolutional networks. However, in our simulations and in its base configuration, QuantGAN does not perform better than our own architecture in terms of empirical coverage, and thus we have left it out of this section for brevity. Wiese et al. (2020) also provide results on QuantGAN and autocorrelation functions.

autoregressive parameter $\phi = 0, 0.25, 0.5, 0.6, 0.7, 0.8, 0.9$. The GB is run for 1000 replications and the CBB is run for 5000 replications. The CBB resamples from the same initial sample as is used to train the GB. We consider CBB block sizes $b_1 = 50, 100, 150$. The GB training block size is $b_1 = 150$, and we use complete sampling with $b_2 = 1000$. The number of replications for GB is lower as the simulation time is considerably higher than for CBB. A GB replication takes around 20–30 min while it is less than a minute for CBB. It is important to note that, in both the CBB and GB, we do not specify the dynamics of the true DGP. The GB assumes that the dynamics can be approximated by some functions of the noise vectors but these functions are not fully specified.

Before presenting the simulation results, we outline the hyper parameters and training of the GAN below.

GAN implementation details. We discuss the hyper parameters and network design of the GAN.

The discriminator has 6 temporal convolution layers with common kernel size 2 and dilations $(1, 2, 4, 8, 16)$. The filters are $(8, 16, 32, 32, 64, 64)$. The output from temporal layers number 1, 2, and 6 are run through adaptive max pooling (AMP) with feature size 16 and concatenated into a feature vector of size 48. This is followed by two fully connected layers that regress into a single output unit. All layers use leaky ReLU activation (Maas, Hannun, & Ng, 2013) except for the final layer which has no activation function. The generator has 6 temporal convolution layers that directly outputs a sample path. The filters are $(128, 64, 32, 32, 16, 1)$. Except for the last layer, all layers use the Tan activation function. The total number of (trainable) discriminator parameters is 233,609 while the generator has 89,921 (trainable) parameters.

The generator noise is sampled from a multivariate standard normal distribution with an identity variance–covariance matrix. To generate a sample path of length b we need $(b + p)$ noise terms where p is the receptive field size in the TC layers and b is either b_1 or b_2 corresponding to the training or sampling stage. The dimension of the noise term is a hyper parameter and can be chosen arbitrarily, in our simulations we use 256. If we stack all the noise terms needed to produce a sample path of size b then we obtain a $(b + p) \times 256$ matrix with iid standard normal distributed entries.

The GAN is trained for 5000 steps based on a single sample from the DGP. We do not employ any (early) stopping criterion, so the GAN is always trained till the final step.[5] The training involves iteratively minimising the batch losses, see Eqs. (D1)–(G1). We use the Adam algorithm for optimisation (Kingma & Ba, 2016). Table A.2 in the Appendix lists all hyper parameters for the GAN in this paper.

4.1. ACF and PACF properties of the GAN samples

We compare the samples produced by the GAN and CBB against the theoretical properties of the AR(1) process.[6] A common discussion is whether the GAN has learned to produce new samples or if it simply reproduces the original samples perfectly. If the generative model learned to perfectly replicate the original sample then the method would perform approximately on-par with CBB. Favourable bootstrapping characteristics of the GAN relative to CBB could indicate that the GAN has an advantage in capturing the dynamics of the DGP and that it does not simply replicate blocks of the original sample.

The theoretical autocorrelation function (ACF) for our AR(1) process is given by $\gamma(j) \equiv \text{Cor}(y_t, y_{t-j}) = \phi^j$ for $\phi = 0.5, 0.8, 0.9$. We estimate the ACF using re-samples from the CBB, and using generated samples from the GB under the complete sampling scheme. The ACF estimates are averaged over 1000 replications. Fig. 2 plots the estimated ACF (full black lines) against the theoretical ACF (dashed black lines), for both the CBB (left column) and GB (right column). In Fig. 2, we have also included the interquartile range (IQR) for the theoretical ACF (blue ribbon), and for the ACF estimated across the 1000 replications for, respectively, the GB (red ribbon) and the CBB (green ribbon). If the GAN in the GB has learned the dynamics of the AR(1) process, then the estimated and the theoretical ACF should be similar, and the theoretical and the estimated IQR should be overlapping. Clearly, for higher values of the autoregressive parameter ϕ, the persistency of the process is stronger and this challenges the GAN to learn longer range dependencies.

From Fig. 2, the estimated ACFs are close to their theoretical counterparts for all lags when $\phi = 0.5$. For $\phi = 0.8$, there is a small upwards bias in the estimated ACFs that is larger for the CBB, particularly, for the intermediate range of lag lengths, i.e., lags 5–20. For $\phi = 0.9$, there is a small but noticeable bias in the estimated ACFs for all lags considered for both GB and CBB. The bias is again uniformly larger for the CBB. Importantly, all theoretical ACFs are well within the IQR of the estimated ACFs.

For $\phi = 0.5$, the estimated IQR of the CBB (green ribbon) is almost identical to the theoretical IRQ (blue ribbon). However, for $\phi = 0.8$ and $\phi = 0.9$, the upper limits of the estimated IQRs for CBB seem to be considerably downward biased for all lags. Noticeably, the estimated IQRs for GB (red ribbon) appear to be much less sensitive to the value of ϕ and the estimated IQSs for GB are only marginally wider than the theoretical IQRs. We find these results encouraging.

Next, we turn to the partial autocorrelation function (PACF). For an AR(p) process the PACF is zero for lags larger than p. Hence, the AR(1) process is expected to have PACF equal to ϕ at lag 1 and zero PACF for all following lags. Fig. 3 depicts the estimated PACF using the GAN samples and plots it against the theoretical PACF at lag 1 (horizontal dotted line). The black horizontal marks denote the estimated IQRs. The estimated PACFs have the expected behaviour for $\phi = 0.5, 0.8$ with values close to, respectively, 0.5 and 0.8 at lag 1, and with values close to zero for all remaining higher order lags. For the highly persistent case $\phi = 0.9$, the estimated PACF is slightly more imprecise with a notable non-zero PACF at lag 2. However, overall, the PACF of the GAN samples clearly suggests that the underlying time series under consideration is a highly persistent AR(1) process.

4.2. Bootstrapping the least-squares estimator

We apply the GAN for re-sampling and examine if it recovers higher-level statistics, in particular, the sampling distribution of the usual least-squares (LS) estimator $\hat{\phi}_{LS}$ of the autoregressive parameter ϕ. The simulation design is identical to that initially described. We obtain 10,000 sample paths from the GAN across 1000 replications with ϕ = (0, 0.25, 0.5, 0.6, 0.7, 0.8, 0.9). For each replication, the GB confidence intervals are constructed as in Section 3.

Fig. 4 contains the simulation results for the GB using complete sampling ($b_2 = 1000$). For values of ϕ in the range $(0, 0.7]$, the GB in general produces better empirical coverage than CBB for all nominal confidence levels, but in particularly for the levels 0.99, 0.95, and 0.9. For the highly persistent processes, none of the re-sampling methods considered have good empirical coverage.

[5] We note that an early-stopping criterion could be implemented based on matching of relevant descriptive statistics and moments between the original sample and those generated by the GAN, such a scheme is used by, e.g., Wiese et al. (2020).

[6] For the implementation of the CBB, we used the Python library by Sheppard (2020).

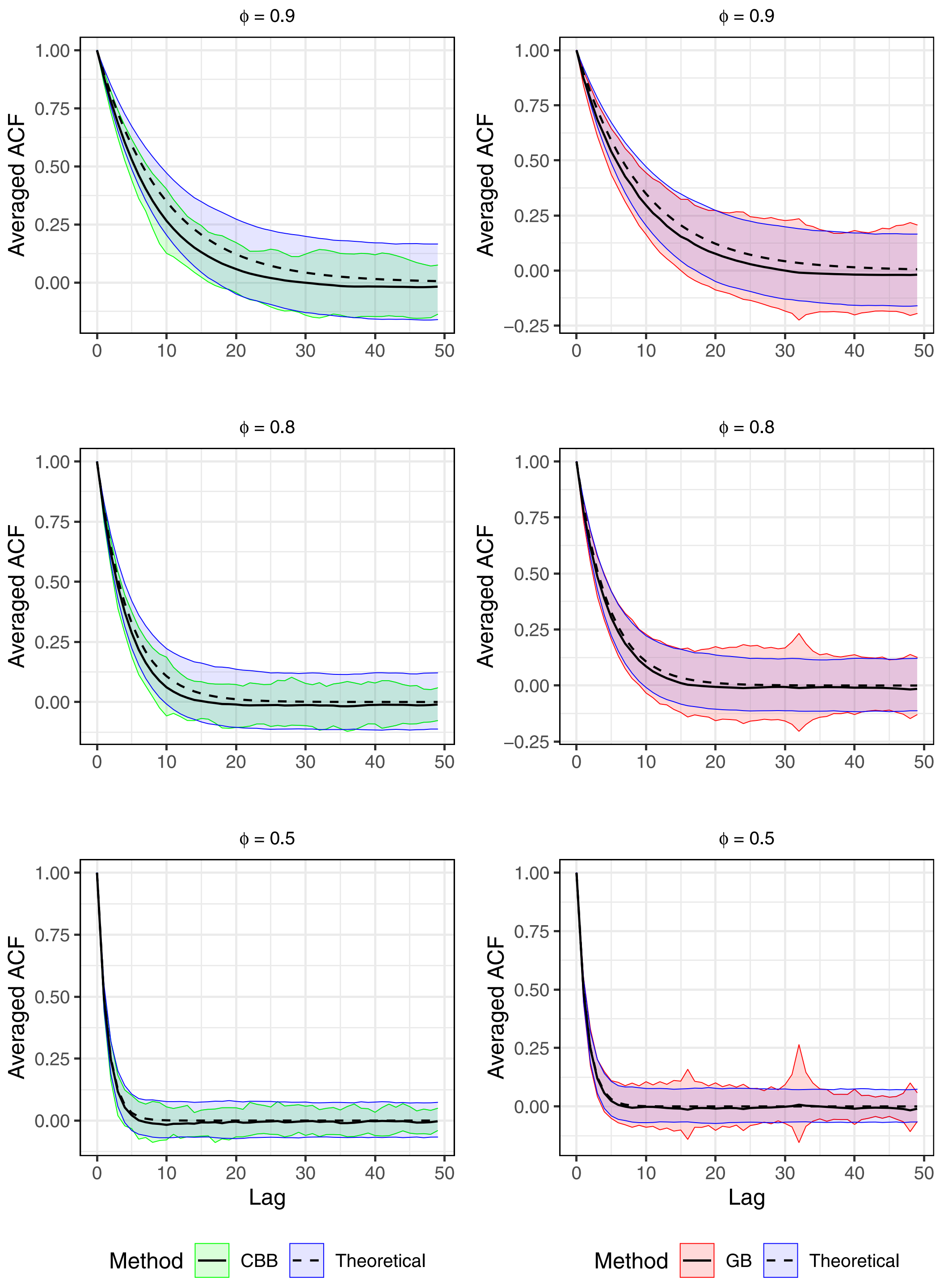


Fig. 2. Theoretical (dashed line, blue ribbon) and sample autocorrelation functions for CBB (green ribbon) and GAN (red ribbon) resamples when $\phi = 0.5, 0.8, 0.9$. Block size for CBB is 150 and training block size for GAN is 150. The ACF estimates and confidence bands are based on 1000 replications of the generative model with 10,000 samples per replication. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

In these cases, the empirical coverage of the GB is at par with the CBB with a block size equal to 100. Not surprisingly, the CBB with the largest block size of 150 has the best coverage.

It is likely that the performance of the GB for highly persistent processes could be improved by increasing the number of layers in the temporal convolutional network. Recall, that a temporal convolutional network with d layers and fixed kernel size 2 accounts for at-most $2^{d+1} - 1$ lags (the receptive field), so networks with larger d can accommodate more persistent processes. How to select the optimal number of layers d, for example, as a function of the persistence of the original process, is ongoing work.

5. Empirical illustration

We now present an empirical illustration of the proposed GB where we consider the estimation of the Sharpe ratio (SR). This is the ratio of expected excess return relative to its return volatility. Being a measure of return per unit of risk, the SR is widely used in financial performance and risk management. Lo (2002) argues

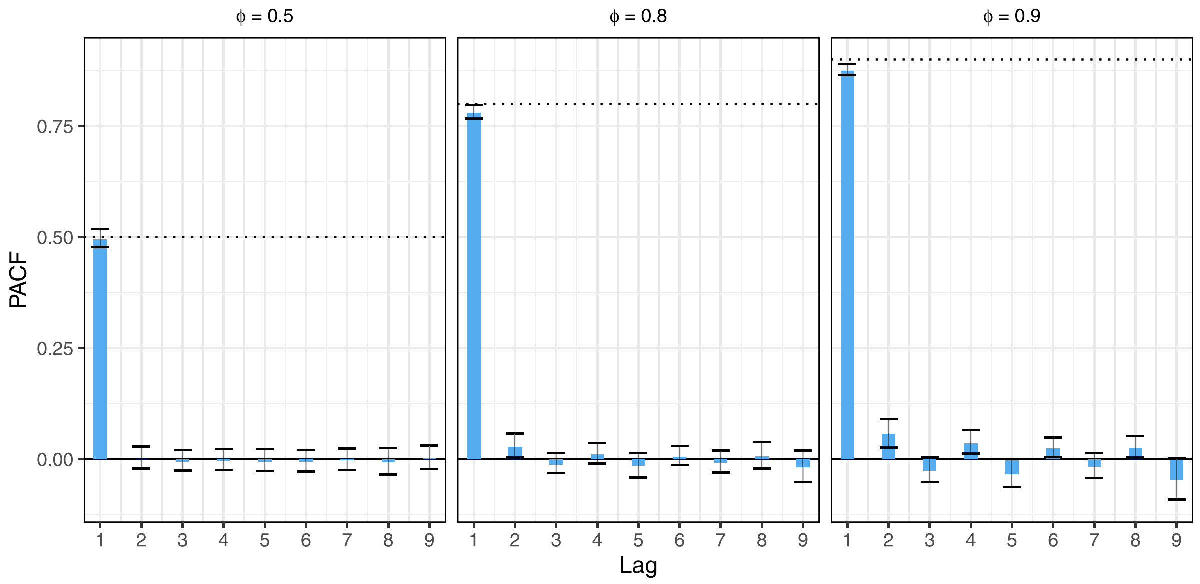


Fig. 3. Sample partial autocorrelation function (blue bars) for $\phi = 0.5$, $\phi = 0.8$ and $\phi = 0.9$. The estimates are based on 1000 replications of the generative model with 10,000 samples per replication. The horizontal black marks depict the interquartile range (IQR) of the estimates across the 1000 replications.

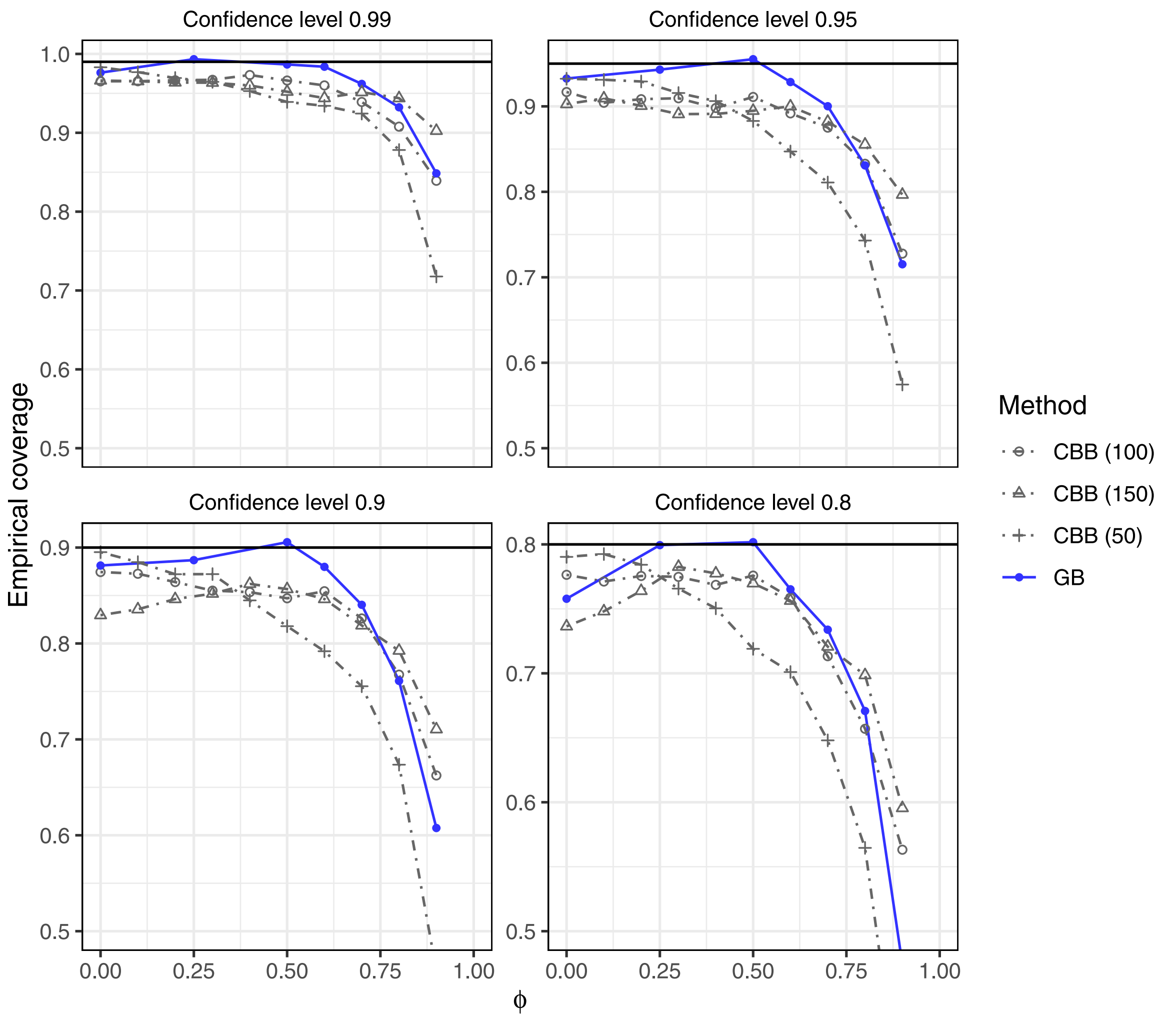


Fig. 4. Empirical coverage of percentile confidence intervals – with nominal confidence levels (0.99, 0.95, 0.90, 0.80) – for the CBB and the GB (using complete sampling) under different choices of the autoregressive parameter ϕ. The horizontal lines depict the corresponding desired nominal confidence levels.

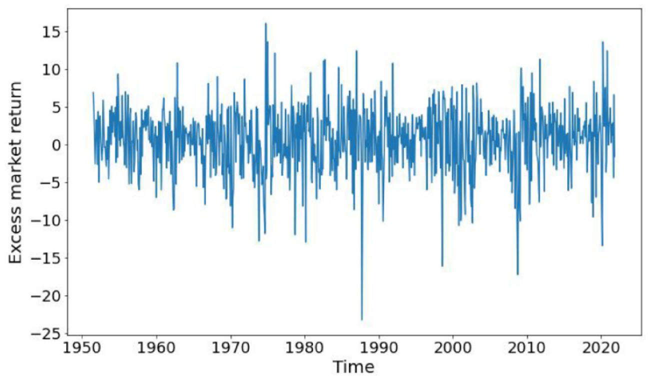


Fig. 5. U.S equity excess market return for the period July 1951–November 2021. The figure plots data from the Fama–French Data Archive.

Table 1
Estimated Sharpe ratio, expected excess market returns and excess market volatility and 95% confidence bands.

	GB		
	Our GAN	QuantGAN	CBB
Sharpe Ratio	0.500	0.478	0.532
$\text{CI}_{0.025}$	0.390	0.345	0.252
$\text{CI}_{0.975}$	0.608	0.618	0.792
Expected excess returns, $\widehat{\mu}$	7.081	7.542	7.925
$\text{CI}_{0.025}$	5.757	5.539	4.328
$\text{CI}_{0.975}$	8.373	9.569	11.644
Excess market volatility, $\widehat{\sigma}$	14.182	15.755	14.899
$\text{CI}_{0.025}$	12.561	14.894	13.568
$\text{CI}_{0.975}$	15.941	16.648	16.319

that the statistical properties of the SR is heavily influenced by the properties of the underlying time series process, and that it is rarely justifiable to assume that returns are iid nor that the return volatility is constant, see, e.g., Lo and MacKinlay (1999). Using a time series of U.S. equity data, we illustrate how to estimate the distribution of the SR statistic using our proposed GB, and how to construct relevant confidence intervals. We benchmark all results against the CBB.

We obtain data on the monthly U.S equity market excess returns from the Fama–French data library.[7] The data covers the period July 1951 to November 2021. Fig. 5 plots the excess returns, while Fig. 6 shows the ACF of the returns (left panel) and the squared returns (right panel). The plots confirm that the time series is non-iid and that it exhibits significant autocorrelation.

In this empirical example, we illustrate the GB using two different GAN architectures. First, as we did in the simulations, we use a GAN with our own TC-architecture. Second, we use the recently proposed QuantGAN (Wiese et al., 2020) as an additional benchmark. The models are trained as described in Section 4 (our GAN) and Appendix A.1 (QuantGAN). The hyper parameters are listed in, respectively, Tables A.2 and A.1. Both models are trained for 20,000 steps or, for QuantGAN, until the early-stopping criterion is satisfied. Based on the trained models, we generate 20,000 synthetic time series sample paths from the GANs' generators.

We first analyse how well the GANs and the CBB can match the relevant sample moments, as these directly affect the sample distribution of the SR. The sample moments include the mean, standard deviation, skewness, kurtosis, as well as the ACF, see, e.g., Lo (2002). In the following, we use "synthetic data" to refer to data resampled from the GANs.

For brevity, we report the autocorrelation coefficients in Fig. A.2 in Appendix A.3. The figure shows the median autocorrelation coefficients and 95% two-sided confidence bands based on resampled data from, respectively, our GAN (red), QuantGAN (blue) and the CBB (cyan).[8] We also report the estimated autocorrelation coefficients, and their 95% asymptotic confidence intervals, based on the actual excess market return data (green), and we show histograms of the estimated autocorrelation coefficients based on resamples from our GAN (red) and the CBB (cyan). For the GANs, the median autocorrelation coefficients are close to the autocorrelation estimates in the observed data. However, for lag 2, the medians for both GANs are positive, while the autocorrelation coefficient in the observed data is negative. For the CBB, the median autocorrelation coefficients appear to be consistently biased towards zero. The confidence bands based on the GANs and the CBB are all wider than those based on the asymptotic distribution of the autocorrelation estimator.

In Fig. A.2 in Appendix A.3, we repeat the computations for the *squared* excess market returns. We find that our own GAN matches the observed autocorrelations well, and does this either at-par, or better than, the QuantGAN. The CBB also matches the observed autocorrelations well. Relative to the CBB, our GAN performs better for lag 3 and 4, while it performs worse on lag 1 and 2. The distributions of autocorrelations based on the resamples from the CBB and our GAN are visibly different, and our GAN produces a distribution with a softer peak than the CBB.

In Figs. 7, we illustrate how well our GAN and the QuantGAN match the first four moments of the observed excess market returns. In Panel (a), the first moment of the synthetic data for both GANs are smaller than in the actual sample. In Panel (b), for our GAN, the estimated standard deviation of the synthetic data is also smaller. However, the differences are small and well within the confidence bands of the distribution. For skewness and kurtosis in Panel (c) and (d), both GANs match well with the observed data, and our GAN performs slightly better on these moments relative to QuantGAN.

In Fig. 8, we plot the histograms of the estimated SRs based on data from, respectively, our GAN (red), QuantGAN (blue), and the CBB (cyan). The three histograms show clear differences. These differences are perhaps somewhat surprising based on the visual similarities of the estimated moments, as discussed earlier. Relative to the GANs, the CBB histogram of SRs suggests a higher value for the SR, and, importantly, it also suggests that the estimation error associated with the SR is higher. The CBB would thus produce wider confidence bands for the SR compared to the two GANs. Table 1 shows the median and the two-sided 95% confidence bands for the SR and for the mean and volatility of the excess market returns, i.e., the numerator and denominator of the SR. We see that the GANs predict a smaller expected excess market return compared to the CBB, and also a lower estimation error (i.e., the confidence bands are narrower). Our GAN also predicts a lower volatility for the excess market returns compared to the CBB, while all methods predict an estimation error for the volatility of approximately the same magnitude (i.e., the confidence bands have similar length).

6. Concluding remarks

We find that GANs are promising for simulating time series data. Our simulation results suggest that GANs can accurately

[7] See the Fama–French data library at https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html (Accessed January 5th, 2022).

[8] In this section, all the confidence bands based on synthetic samples are computed by the percentile bootstrap approach, see, e.g., Davison and Hinkley (1997).

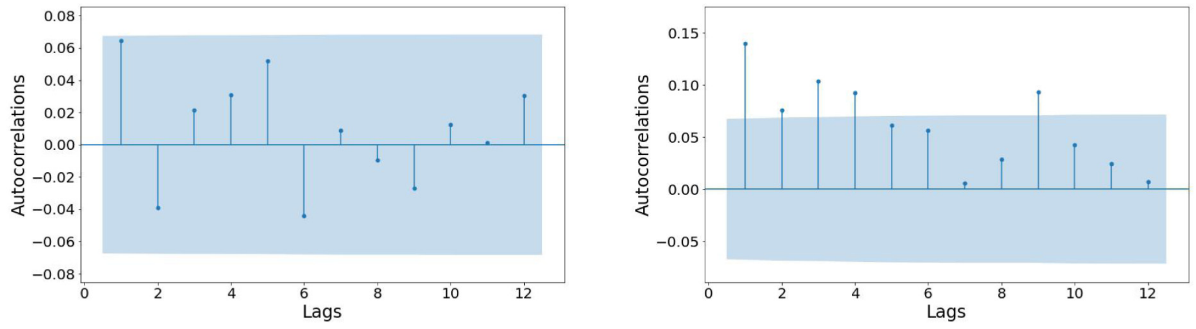


Fig. 6. Autocorrelation functions estimated in the observed data. Left: Excess market returns. Right: Excess market returns squared. Shaded areas indicate "asymptotic" 95% confidence bands.

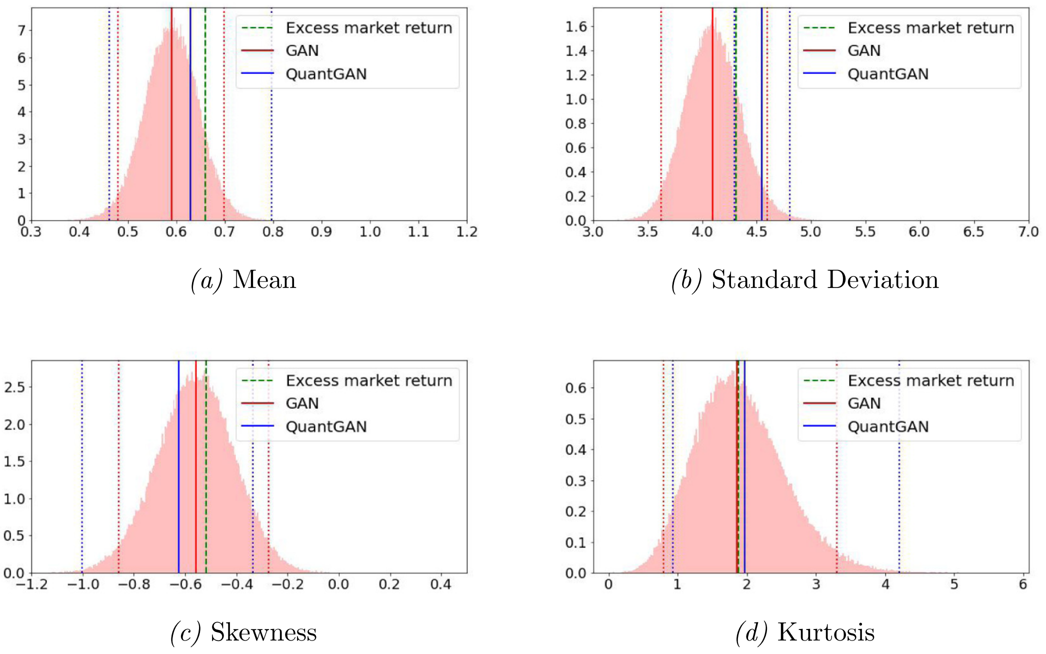


Fig. 7. The first four moments estimated using data from, respectively, our GAN (red), QuantGAN (blue), and the observed data (dashed green). We also show the histogram of the estimated moments based on data from our GAN (red). The solid lines indicate the median estimate, and the dotted lines show a corresponding 0.95 confidence band. The green dashed line shows the estimated moment based on the observed data. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

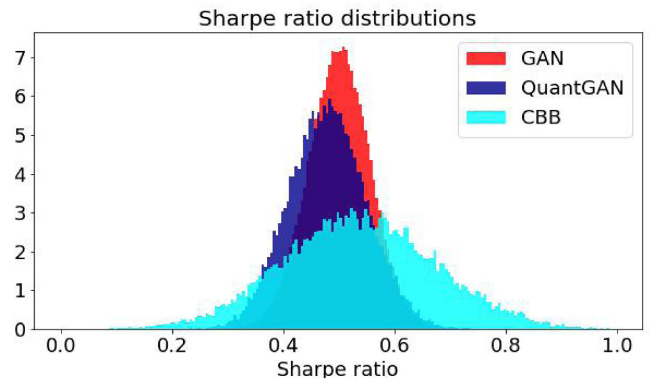


Fig. 8. The histograms show the estimated SRs using resamples from, respectively, our GAN (red), QuantGAN (blue), and the CBB (cyan). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table A.1
Generative Bootstrap (GB) QuantGAN hyper parameters.

Hyper parameter	Value
Discriminator learning rate, lr_d	0.0003
Generator learning rate, lr_d	0.0001
Gradient penalty, λ	0
Batch size, n_b	512
Discriminator init updates, N_{init}	1
Discriminator updates, $N_{discriminator}$	1
Generator updates, $N_{generator}$	1
Hidden Dimensions	50, 50, 50, 50, 50, 50
Hidden Dimensions Skip	50
Latent dimension	3
Block Size	127
Receptive Field Size	127
ACF loss threshold	0.3
ACF loss lags	64
Adam optimiser, ϵ	10^{-8}
Adam optimiser, β_1	0.0
Adam optimiser, β_2	0.9

learn the dynamics of common autoregressive time series processes using temporal convolutional networks. In addition, when GANs are used in our proposed GB, it is compelling that they appear to improve the empirical coverage relative to the CBB.

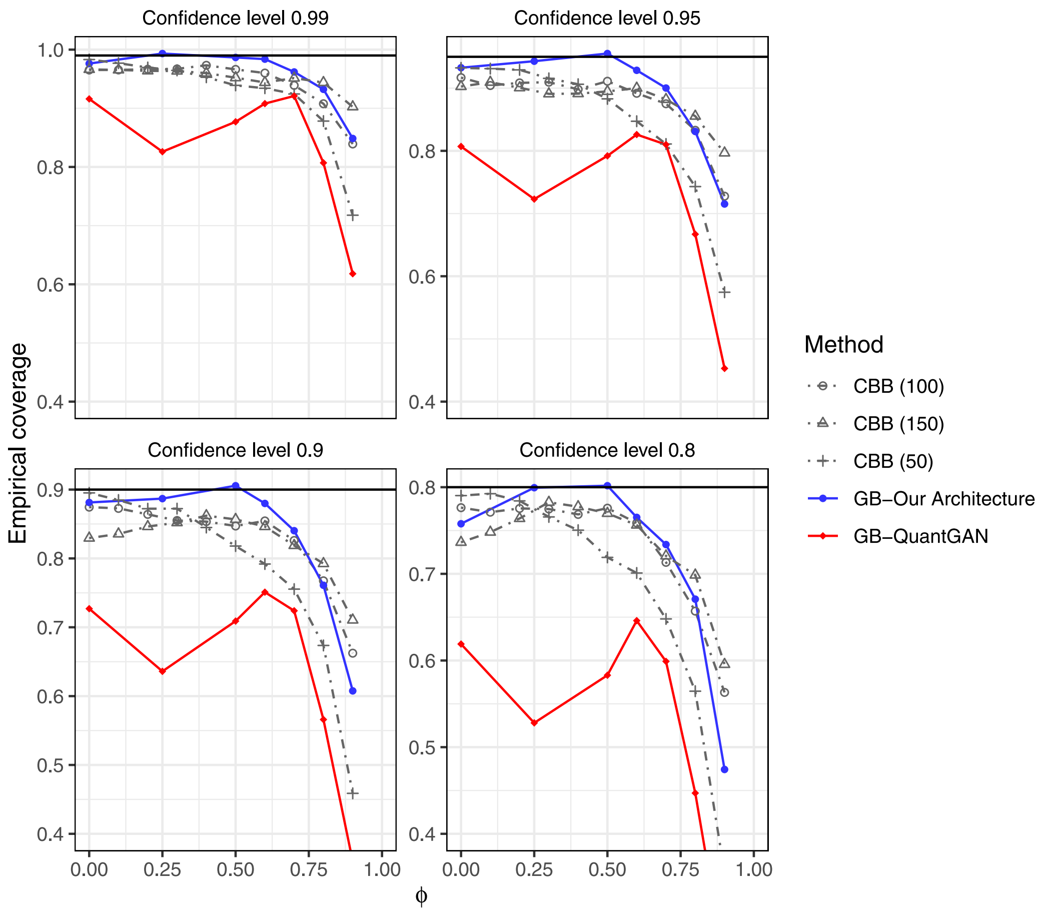


Fig. A.1. Empirical coverage of percentile confidence intervals – with nominal confidence levels (0.99, 0.95, 0.90, 0.80) – for the CBB and the GB (both our architecture, and the QuantGAN architecture) under different choices of the autoregressive parameter ϕ. The horizontal lines depict the corresponding desired nominal confidence levels.

Table A.2
Hyper parameters used for our GAN architecture.

Hyper parameter	Value
Discriminator learning rate, lr_d	0.00025
Generator learning rate, lr_d	0.00025
Gradient penalty, λ	20
Batch size, n_b	64
Discriminator init updates, N_{init}	50
Discriminator updates, $N_{discriminator}$	5
Generator updates, $N_{generator}$	1
Initial weight distribution	N(0, 0.02)
Adam optimiser, ϵ	10^{-8}
Adam optimiser, β_1	0.5
Adam optimiser, β_2	0.9

This gives credibility to the use of GANs on data from time series processes that are unknown, and it suggests that more powerful bootstraps can potentially be developed by relying on GANs for re-sampling. Our empirical illustration shows that our GAN can generate synthetic time series with ACFs that match those of the excess market return series. Using samples from the GAN, we compute the SR and estimate its moments up to the 4th order, and we find that they are similar to those based on the observed excess market returns. Finally, we use the trained GAN in our proposed GB to bootstrap the SR and find that this produces 95% confidence bands that are somewhat narrower compared to the corresponding confidence bands based on the CBB.

It is important to note that the various bootstraps for dependent data have theoretical justifications and their properties have been theoretically derived, see, e.g., the overview in Lahiri (1999). The GAN and GB currently do not have this theoretical reassurance.

The GAN relies on a large number of hyper parameters and design choices. We have not systematically investigated how sensitive our results are to these choices, and more research is required to determine this. We have used sensible defaults for batch sizes, learning rates, gradient penalty, update iteration scheme, activation functions, and optimiser, but these are by no means optimal choices. This is a general shortcoming in the GAN literature and little is known about how to optimally choose these values.

Our simulations rely on a simple and basic time series process, the AR(1). It would be fruitful to consider the performance on more general stationary processes (ARMA) and in settings where the error term has stochastic variance, e.g., GARCH. Such stochastic variance, unsurprisingly, seems to be present in our empirical illustration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

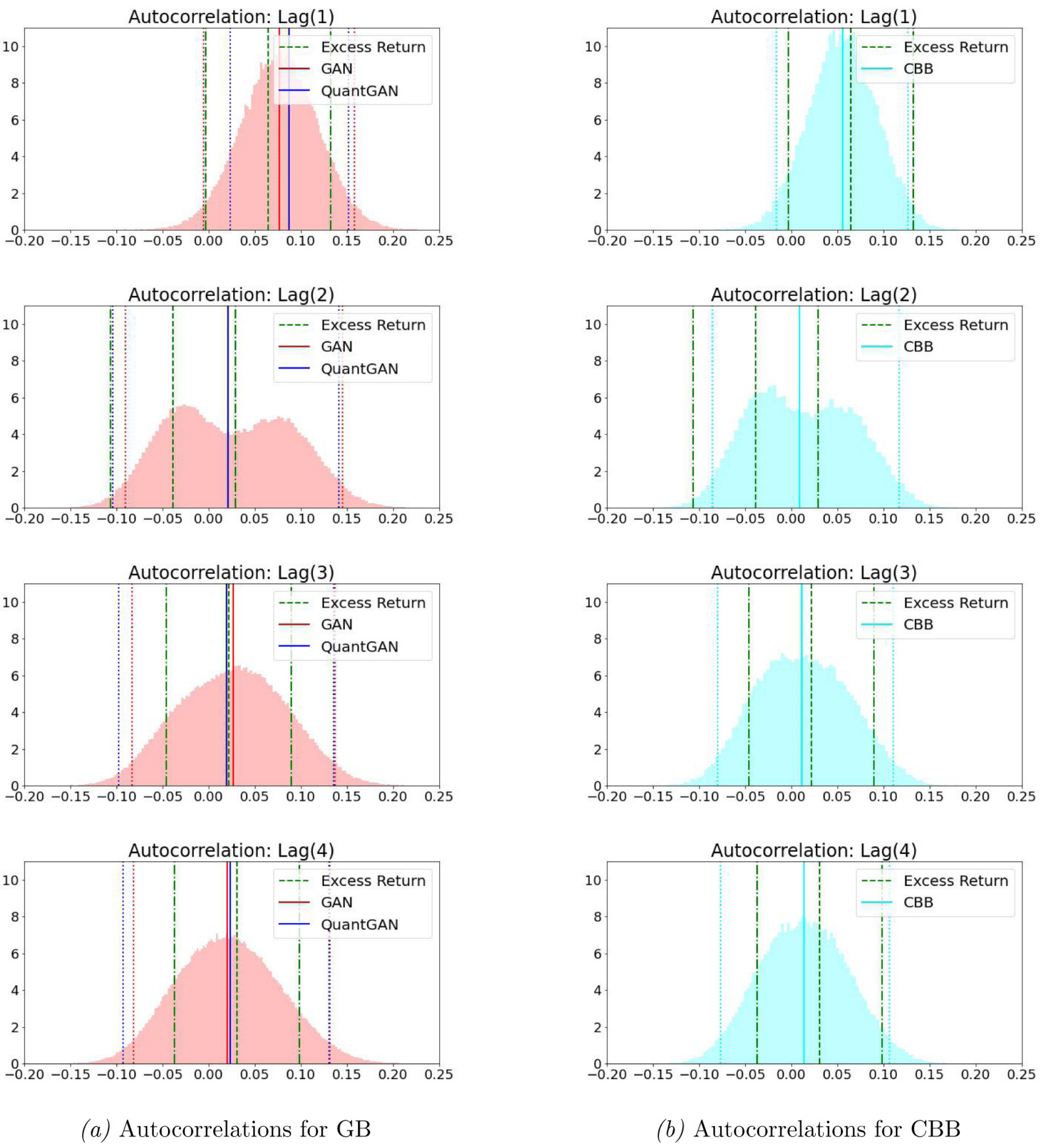


Fig. A.2. Autocorrelation coefficients for the excess returns. Column (a) shows the median autocorrelation coefficients for GB when using QuantGAN (blue lines) or our own GAN (red lines) for resampling. The corresponding 0.95 confidence bands are shown with dotted lines. The background shows, in red, the histogram of the estimated autocorrelation coefficients for resamples from our own GAN. Column (b) shows the median autocorrelation coefficients based on data resampled from the CBB (cyan lines). The corresponding 0.95 confidence band is shown with dotted lines. The background shows, in cyan, the histogram of the estimated autocorrelation coefficients for resamples from the CBB. The estimated autocorrelation coefficients and their asymptotic 0.95 confidence band, based on the observed data, are shown with, respectively, green dashed and green dash-dotted lines. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Appendix

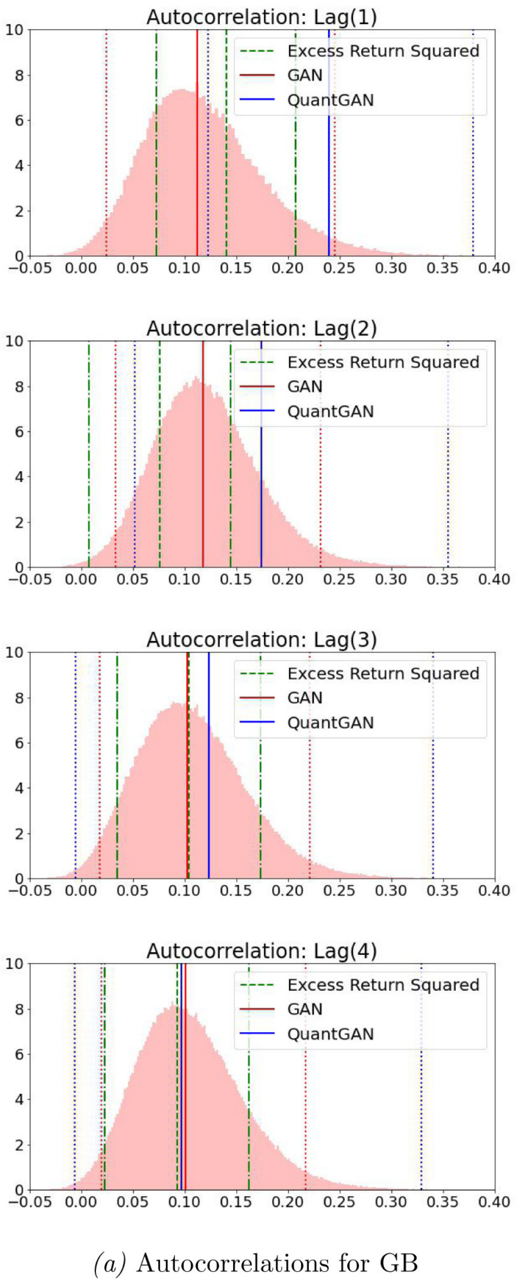
A.1. Using QuantGAN as the backbone of the GB

As mentioned in the main text, we can use any suitable GAN in the GB. We explore this here, by comparing the GB when using, respectively, QuantGAN (Wiese et al., 2020) and our own GAN. The simulation design, and the training of our own GAN, are exactly as described in Section 4 of the main text. We report our hyperparameters for QuantGAN in Table A.1. We train QuantGAN for a maximum of 2000 steps, but stop the training early based on moment-like conditions. We refer to Wiese et al. (2020) for more details.

Fig. A.1 replicates Fig. 4 from the main text, but adds an additional red line that shows the empirical coverage from the GB when using QuantGAN (GB-QuantGAN). We see that GB-QuantGAN performs, at best, at-par with CBB, and in most cases worse.

A.2. Hyper parameters — our GAN

See Table A.2.



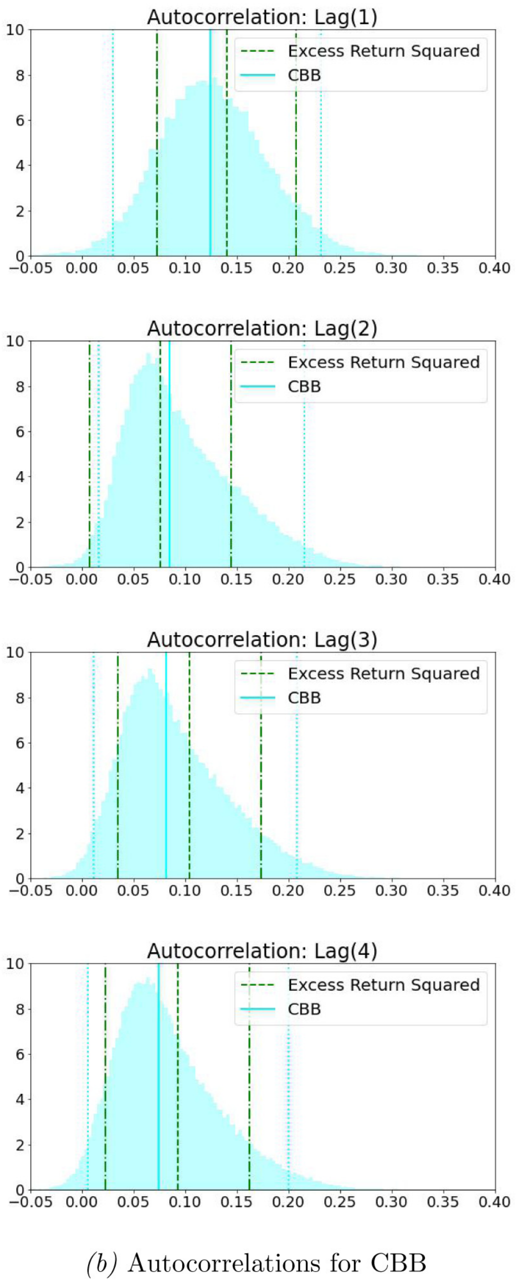


Fig. A.3. Autocorrelation coefficients for the squared excess returns. Column (a) shows the median autocorrelation coefficients for GB when using QuantGAN (blue lines) or our own GAN (red lines) for resampling. The corresponding 0.95 confidence bands are shown with dotted lines. The background shows, in red, the histogram of the estimated autocorrelation coefficients for resamples from our own GAN. Column (b) shows the median autocorrelation coefficients based on data resampled from the CBB (cyan lines). The corresponding 0.95 confidence band is shown with dotted lines. The background shows, in cyan, the histogram of the estimated autocorrelation coefficients for resamples from the CBB. The estimated autocorrelation coefficients and their asymptotic 0.95 confidence band, based on the observed data, are shown with, respectively, green dashed and green dash-dotted lines. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

A.3. Additional illustrations

See Figs. A.1–A.3.

References

Arjovsky, Martin, Chintala, Soumith, & Bottou, Léon (2017). Wasserstein generative adversarial networks. In Doina Precup, & Yee Whye Teh (Eds.), *Proceedings of machine learning research: Vol. 70, Proceedings of the 34th international conference on machine learning* (pp. 214–223). PMLR.

Athey, Susan, Imbens, Guido W., Metzger, Jonas, & Munro, Evan (2021). Using Wasserstein generative adversarial networks for the design of Monte Carlo simulations. *Journal of Econometrics*, [ISSN: 0304-4076] http://dx.doi.org/10.1016/j.jeconom.2020.09.013.

Biau, Gérard, Cadre, Benoît, Sangnier, Maxime, & Tanielian, Ugo (2018). Some theoretical properties of GANs. *The Annals of Statistics*, *48*, http://dx.doi.org/10.1214/19-AOS1858.

Biau, Gérard, Sangnier, Maxime, & Tanielian, Ugo (2021). Some theoretical insights into Wasserstein GANs. *Journal of Machine Learning Research*, *22*(290), 1–45.

Borovykh, Anastasia, Bohte, Sander, & Oosterlee, Cornelis W. (2017). Conditional time series forecasting with convolutional neural networks. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*: *Vol. 10614*, (pp. 729–730).

Brophy, Eoin, Wang, Zhengwei, She, Qi, & Ward, Tomás (2021). Generative adversarial networks in time series: A survey and taxonomy. *CoRR*, abs/2107.11098.

Davison, A. C., & Hinkley, D. V. (1997). *Cambridge series in statistical and probabilistic mathematics*, *Bootstrap methods and their application*. Cambridge University Press, http://dx.doi.org/10.1017/CBO9780511802843.

Efron, Bradley (1981). Nonparametric standard errors and confidence intervals. *The Canadian Journal of Statistics*, (la Revue Canadienne de Statistique) *9*(2), 139–158.

Efron, Bradley (1987). Better bootstrap confidence intervals. *Journal of the American Statistical Association*, *82*(397), 171–185.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, et al. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Gui, Jie, Sun, Zhenan, Wen, Yonggang, Tao, Dacheng, & Ye, Jieping (2020). A review on generative adversarial networks: Algorithms, theory, and applications. *CoRR*, abs/2001.06937.

Gulrajani, Ishaan, Ahmed, Faruk, Arjovsky, Martin, Dumoulin, Vincent, & Courville, Aaron C. (2017). Improved training of Wasserstein GANs. In *Advances in neural information processing systems* (pp. 5767–5777).

Haas, Moritz, & Richter, Stefan (2020). Statistical analysis of Wasserstein GANs with applications to time series forecasting. arXiv preprint 2011.03074.

Hyland, Stephanie L., Esteban, Cristóbal, & Rätsch, Gunnar (2017). Real-valued (medical) time series generation with recurrent conditional GANs. arXiv preprint 1706.02633.

Kaji, Tetsuya, Manresa, Elena, & Pouliot, Guillaume (2018). *Deep inference: Artificial intelligence for structural estimation*: *Tech. rep.*.

Kingma, Diederik P., & Ba, Jimmy (2016). Adam: A method for stochastic optimization. In *Proceedings of the 3rd international conference on learning representations*.

Koshiyama, Adriano, Firoozye, Nick, & Treleaven, Philip (2021). Generative adversarial networks for financial trading strategies fine-tuning and combination. *Quantitative Finance*, *21*(5), 797–813. http://dx.doi.org/10.1080/14697688.2020.1790635.

Kunsch, Hans R. (1989). The jackknife and the bootstrap for general stationary observations. *The Annals of Statistics*, 1217–1241.

Lahiri, Soumendra N. (1999). Theoretical comparisons of block bootstrap methods. *The Annals of Statistics*, 386–404.

Liu, Regina Y., Singh, Kesar, et al. (1992). Moving blocks jackknife and bootstrap capture weak dependence. *Exploring the Limits of Bootstrap*, *225*, 248.

Lo, Andrew W. (2002). The statistics of sharpe ratios. *Financial Analysts Journal*, *58*(4), 36–52. http://dx.doi.org/10.2469/faj.v58.n4.2453.

Lo, Andrew W., & MacKinlay, A. Craig (1999). *A non-random walk down wall street*. Princeton University Press.

Maas, Andrew L., Hannun, Awni Y., & Ng, Andrew Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. Icml*: *Vol. 30*, (1).

Ni, Hao, Szpruch, Lukasz, Vidales, Marc Sabate, Xiao, Baoren, Wiese, Magnus, & Liao, Shujian (2021). Sig-Wasserstein GANs for time series generation. *CoRR*, abs/2111.01207.

van den Oord, Aaron, Dieleman, Sander, Zen, Heiga, Simonyan, Karen, Vinyals, Oriol, Graves, Alex, et al. (2016). WaveNet: A generative model for raw audio. In *Proc. 9th ISCA workshop on speech synthesis workshop* (p. 125).

Paparoditis, Efstathios, & Politis, Dimitris N. (2001). Tapered block bootstrap. *Biometrika*, *88*(4), 1105–1119.

Politis, Dimitris N., & Romano, Joseph P. (1992). A circular block-resampling procedure for stationary data. *Exploring the Limits of Bootstrap*, *2635270*.

Salimans, Tim, Goodfellow, Ian J., Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, & Chen, Xi (2016). Improved techniques for training GANs. *CoRR*, abs/1606.03498.

Sen, Rajat, Yu, Hsiang-Fu, & Dhillon, Inderjit S. (2019). Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems*: *Vol. 32*, Curran Associates, Inc..

Shao, Xiaofeng (2010). The dependent wild bootstrap. *Journal of the American Statistical Association*, *105*(489), 218–235.

Sheppard, Kevin (2020). bashtage/arch: Release 4.15 (Version 4.15). Zenodo.

Smith, Kaleb E., & Smith, Anthony O. (2020). Conditional GAN for timeseries generation. *CoRR*, abs/2006.16477.

Sun, He, Deng, Zhun, Chen, Hui, & Parkes, David C. (2020). Decision-aware conditional GANs for time series data. *CoRR*, abs/2009.12682.

Wiese, Magnus, Knobloch, Robert, Korn, Ralf, & Kretschmer, Peter (2020). Quant GANs: Deep generation of financial time series. *Quantitative Finance*, 1–22.

Wold, Herman (1939). A study in analysis of stationary time series. *Journal of the Royal Statistical Society*, *102*(2), 295–298. http://dx.doi.org/10.1111/j.2397-2335.1939.tb01266.x.

Yoon, Jinsung, Jarrett, Daniel, & van der Schaar, Mihaela (2019). Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems*: *Vol. 32*, Curran Associates, Inc..

Yu, Fisher, & Koltun, Vladlen (2016). Multi-scale context aggregation by dilated convolutions. *CoRR, abs/1511.07122*.