Combined Estimation of Degradation and Soiling Losses in Photovoltaic Systems

Åsmund Skomedal and Michael G. Deceglie

***Abstract*—Being able to quantify energy production losses in photovoltaic (PV) systems is important in order to reduce the risk associated with investing in PV. Two such loss components are degradation rates and soiling losses. However, in systems where both these phenomena exist, quantifying them is not straightforward because of their combined effect on the power output. In this article, we propose an algorithm that iteratively decomposes a performance index time series of a PV system into a soiling component, a degradation component, and a seasonal component. This makes it possible to simultaneously estimate soiling losses and degradation rates of PV systems. Bootstrapping is used to estimate confidence intervals so that both data uncertainty and model uncertainty is taken into account. Based on simulated data we show that this method makes more accurate estimates of soiling losses and degradation rates than relevant state-of-the-art methods.**

***Index Terms*—Data analysis, monitoring, photovoltaics (PV), PV systems, solar power generation, time series analysis.**

I. Introduction

THE accumulation of soil and dirt, also called *soiling*, on photovoltaic (PV) modules blocks incident light and leads to lost energy production. The severity of soiling depends on local soiling deposition rates and the frequency of cleaning or rain. In some locations, because of frequent rainfall, soiling will never build up enough to lead to any measurable production loss, while in other locations soiling can lead to a decrease in performance above 1%/day [1]. In many cases, an active soiling mitigation strategy is economically beneficial. The choice of strategy, whether it is manual cleaning, robotic cleaning, or the application of an antisoiling treatment to the PV module surfaces, depends on the severity of soiling in each location. If one is able to quantify the historic performance losses because of soiling, one is much better placed to select a suitable strategy.

Manuscript received February 12, 2020; revised May 29, 2020 and July 21, 2020; accepted August 17, 2020. Date of publication September 3, 2020; date of current version October 21, 2020. This work was supported in part by the ENERGIX Programme under Project 282404, in part by Research Council of Norway and Industry Partners, in part by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308, and in part by U.S. Department of Energy Office of Energy Efficiency and Renewable Energy (EERE) Solar Energy Technologies Office (SETO) Agreement 34348. *(Corresponding author: Åsmund Skomedal.)*

Åsmund Skomedal is with the Institute for Energy Technology, 2007 Kjeller, Norway (e-mail: asmunds@ife.no).

Michael G. Deceglie is with the National Renewable Energy Laboratory, Golden, CO 80401 USA (e-mail: michael.deceglie@nrel.gov).

Digital Object Identifier 10.1109/JPHOTOV.2020.3018219

Degradation is the decrease in PV system performance over time due to mechanisms not reversed in the relevant field conditions. It is well known that all PV systems suffer from degradation [2]. Being able to identify underperforming systems is important for PV owners, while being able to quantify typical degradation rates on fleet-scale data is important for the field in general, in order to reduce the risk associated with investing in PV. We note that in the present application, unrecovered soiling manifests as degradation and are indistinguishable from other degradation modes without the aid of additional sensors designed to quantify soiling.

Tools for the estimation of both soiling losses [3], [4], and degradation rates [5] directly from PV production data already exist. However, because of the combined effect these phenomena have on PV performance, degradation rates are difficult to extract in systems with a lot of soiling. Indeed, this was duly noted in the article first proposing the year-on-year (YOY) method for the estimation of degradation rates [6], as well as in later work [7]. This challenge has been addressed in a recent publication involving one of the authors of this article [8]. In that study, the degradation estimate was improved by estimating the soiling component and doing the YOY method on a soiling-corrected performance index (*PI*). However, we show that a more accurate estimate of either component can be made if the two components are estimated iteratively, and if a seasonal component is also included in the model.

The tool presented in this work will be published in the PV analytics package for Python, RdTools[1] [9].

II. Method

A. Model Assumptions

According to the IEC 61724-1 standard [10], the *PI* of a PV system is the measured energy output of the system divided by the expected energy. In principle, the method we are proposing is agnostic as to what model is used to estimate the expected energy, as long as it is on a daily resolution. Daily resolution eliminates intraday variations in the *PI*, while still being a sufficiently fine time resolution to adequately capture soiling patterns. We propose that the *PI* can be decomposed into four components, and that these four components have a multiplicative effect on the *PI*. Thus, for each day d

$$PI_d = SR_d \cdot SC_d \cdot D_d \cdot n_d \tag{1}$$

[1]Currently, a version is available to the public at https://github.com/asmunds/rdtools/tree/7592329ef0f6549be5515d4bc04b24452d327739

where $SR = \frac{\text{performance of dirty system}}{\text{performance of clean system}} \in [0, 1]$ is the soiling ratio, SC is the seasonal component, D is the degradation trend, and n is the noise.

The following are assumed for the four components. According to previously reported observations [3], [11]–[13], the soiling signal consists of a repeated pattern of declining segments followed by a sudden positive step (a cleaning event), recovering SR near to 1. We do not assume constant soiling rates between cleaning events, only that it is slowly changing. The seasonal component varies slowly, is centered at 1, and has a period of one year. The degradation trend is linear, starting at 1 at the beginning of the time series. The noise, also referred to as *residuals,* is what remains in the *PI*-signal after dividing out the other components, i.e., $n = \frac{PI}{SR \cdot SC \cdot D}$. The noise is not necessarily centered at 1.

Note that the soiling ratio is typically modeled as a piecewise linear function (giving a piecewise constant soiling rate) with breakpoints at the cleaning events, first suggested (as an approximation) by Kimber *et al*. [11]. This is usually a good approximation that simplifies the modeling, but in many cases the soiling rate changes between the cleaning events. This can be resolved by allowing breakpoints between the cleaning events. However, we take an alternate approach without the assumption of piecewise linearity, instead assuming slowly changing soiling rates between the cleaning events.

B. Iterative Solution

In order to decompose the *PI* into the components in (1), we propose an iterative, self-consistent solution. Each iteration has four steps. In these steps, the algorithm infers cleaning events, and estimates SR, SC, and D, respectively. The soiling component is estimated with a Kalman filter, the seasonal component is estimated with seasonal and trend decomposition using LOESS (STL) [14], and the degradation component is estimated using the YOY method [5]. When the four steps have been done, a new iteration starts, where the estimates made in the previous iteration is used to get more accurate component estimates. This algorithm, which we term *Combined Degradation and Soiling* (CODS), is illustrated in the flow chart in Fig. 1. In the following sections we describe each of the estimation models, followed by some general remarks on the algorithm as a whole.

1) Detection of Cleaning Events: The first step in the CODS algorithm is the detection of cleaning events. Much as in [3], the cleaning events are detected by considering shifts in the centered 9-day rolling median of the input signal. If Δ is the day-to-day change in the rolling median, cleaning events are defined as days where $\Delta > Q3 + a \cdot IQR$. In this case, $Q3$ and IQR are the third quartile and the interquartile range of $|\Delta|$, respectively, and a decides the sensitivity of the cleaning detection. a is one of the tuning parameters of the CODS algorithm. In Section II-C, we will come back to how the tuning parameters are set.

The choice of cleaning detection algorithm was decided by extensive testing on the synthetic data. These data are described in Section II-D. In addition to a range of variants of the chosen method, other methods such as change point detection, total variation filtering, and shift detection with the Kalman filter, were also tested. The chosen method was the one that (in our implementation) gave the highest mean F_1 score based on around 1000 realizations on the synthetic data. The F_1 score is defined as $F_1 = 2 \cdot \frac{P \cdot R}{P + R}$, where $P = \frac{TP}{TP+FP}$ and $R = \frac{TP}{TP+FN}$, and *TP, FP,* and *FN* are the number of true positives, false positives, and false negatives, respectively. Thus, the F_1 score is the harmonic mean of the fraction of inferred cleaning events that are true (*P*), and the fraction of true cleaning events that are detected (*R*).

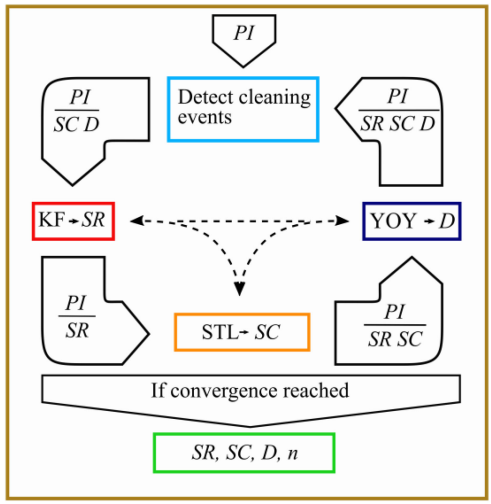

Fig. 1. Flowchart for the CODS algorithm. The algorithm takes a time series of daily PI as input, and iteratively decomposes this into a soiling component SR, a seasonal component SC, and a degradation component D. The input to each step is indicated in the open arrows. On the first iteration, component estimates not yet obtained are assumed to be unity. The dashed arrows in the middle of the chart signify that the order of the steps can be changed, as described in Section II-B.5.

In a dataset coming from the field, we expect missing data. If a cleaning event has happened in a period of missing data, we cannot know which of these days the cleaning event happened. Thus, to account for the effect of missing data on cleaning event detection, the input to this step is either filled with the value of the last datapoint before the period (forward filled), or the value of the first datapoint after the period (backward filled). This makes up two different model choices for the CODS model. During bootstrapping, the CODS algorithm is run several times with different filling modes. In this way, the uncertainty in the date of the cleaning event is captured. We will come back to how this is done in Section II-C.

2) Kalman Filter for Estimating ***SR****:* In short, the Kalman filter is an algorithm that takes a series of noisy measurements and estimates the unknown variables that give rise to these measurements. In our case, the measurements are the daily values of the (corrected) *PI* signal, and the unknown variables are the denoised signal and its rate of change. In a postprocessing step, we estimate the soiling component SR, and its rate of change R_{SR} based on the KF estimate. We note that R_{SR} is not constrained to be piecewise constant as in the typical sawtooth model for PV soiling.

We can look at the KF as a method for removing noise from our signal. The input signal in our case is the PI corrected for seasonality and degradation; $\frac{PI}{SC \cdot D}$, where the last found component estimates for SC and D are used. Component estimates not yet obtained are assumed to be unity.

For each point in time, the KF algorithm combines a prediction of the unknown variables, based on previous estimates, with a measurement (i.e., a datapoint in the input signal), giving a new estimate of the unknown variables. Both the prediction and the measurement have associated uncertainties, and they are weighted by the uncertainties when calculating the new estimate. In this way, the KF combines the measurement with past estimates of the unknown variables, thus giving a better estimate of the unknown variables than if the measurements were considered individually.

We think KF is suitable for estimating *SR* because it is computationally efficient compared with e.g., piecewise linear fitting, and because it allows for slowly varying soiling rates. Another advantage of using KF is that it handles missing data well: If a measurement is missing, the estimate of the unknown variables at this point in time is simply the prediction. Of course, if there is a long period of missing data (e.g., $>$ 1 month), this will lead to large uncertainties in the estimate of the unknown variables. Lastly, it is easy to extend the KF to make predictions of future states, since a prediction step is already an inherent part of the algorithm. For further reading about the Kalman filter, we refer to [15] and [16]. We have used the Python implementation of the KF found in FilterPy [17].

One assumption of KF is that the noise is Gaussian. In addition, the uncertainty in the predicted state is expressed by a Gaussian. Since the product of two Gaussians is another Gaussian, this assumption makes the filter computationally cheap. Of course, the noise of *PI* data from the field will not necessarily be Gaussian. However, in general, the filter performs well even when this assumption is violated, as long as the noise is symmetric [15]. Systematic measurement errors and asymmetric noise, though, may bias the results. We discuss the issue of systematic errors further in Section II-B.5.

The abrupt effect of cleaning events on SR is accounted for through the so-called *control input*. This is an input parameter to the KF algorithm that adjusts the prediction given a known perturbation. This means that, at the detected cleaning events, we will provide input to the algorithm that will allow it to adjust the prediction of the unknown variables according to the abrupt change in the input signal. The cleaning events are detected before running the KF, as described in Section II-B.1, but the magnitude of the control input is decided runtime. The control input is set so that the prediction of the signal value equals the median of the input signal in the seven days following the cleaning event, but the rate of change is not altered.

After the KF has been run (doing a *forward pass*), the unknown variable estimate is smoothed by running an Rauch–Tung–Striebel (RTS)-smoother [18] on the data between the cleaning events. The RTS-smoother is a KF-based method for finding the maximum likelihood estimate of the unknown variables. It can be used when KF is applied to offline data, as it is in our case. It takes advantage of the information in the forward pass, and does a *backward pass*, updating the estimate of the unknown variables one step at a time, starting at the last datapoint, going backward. This way, it combines the KF estimate at each time step with smoothed estimates from the future. Thus, when the RTS-smoother is used, information from both future and past data points is used.

In the next step, we take advantage of the smoothed estimate to prune the cleaning events, removing inferred cleaning events that are unlikely to be real cleaning events. This is done as follows: Let the average estimates of the unknown variable in the week following the cleaning events be denoted by $\overline{x_{ce}}$. False cleaning events are numerical outliers in $\overline{x_{ce}}$, where an outlier is defined as a $\overline{x_{ce}} < Q1 - b \cdot IQR$. Here, $Q1$ and IQR are the first quartile and the interquartile range of $|\overline{x_{ce}}|$, respectively. b decides the sensitivity of the pruning of cleaning events, and is the second tuning parameter of the CODS algorithm.

In order to estimate SR and R_{SR}, a final postprocessing step is necessary. This can be done in two ways, one that assumes perfect cleaning, and one that does not. If perfect cleaning is assumed, in each period between two cleaning events, SR and R_{SR} equals the KF estimates of the unknown variables normalized so that $SR_d = 1$ on the day following the previous cleaning event. If perfect cleaning is not assumed, SR and R_{SR} equals the KF estimate. When the latter approximation is used, *SR* can be estimated to have a value above 1, which is unphysical. In these cases, it is capped at 1. The latter approximation makes sense when sufficiently accurate estimates of SC and D have been found so that SR is the only component remaining in the input signal to KF. The CODS algorithm starts out by assuming perfect cleaning, but drops this assumption after a few iterations, as explained in Section II-B.5 below.

3) STL for Estimating $\boldsymbol{SC}$: Seasonal and trend decomposition using LOESS (STL) is a method for decomposing time series into trend, seasonal, and residual components [14]. It works by sequentially fitting a locally estimated scatterplot smoother (LOESS) to the time series. In our case, we only use the estimated seasonal component. The advantages of using STL, instead of methods such as X-11 decomposition or classical decomposition, is that it is more computationally efficient, it is robust to outliers, it is simple to implement, and it handles daily data with yearly periodicity. In the Python implementation of the CODS algorithm, we use an implementation of the STL from the statistical modeling library Statsmodels [19], [20]. The input signal to the STL model is $\boldsymbol{log}(\frac{\boldsymbol{PI}}{\boldsymbol{SR}})$. The log transform is necessary because the components are multiplicative, and the Statsmodels implementation of STL assumes additive components. Although, the original STL algorithm handles missing data, the current Statsmodels implementation of it does not. We deal with this by doing a linear interpolation between the two data points on each side of the period of missing data.

4) YOY for Estimating $\boldsymbol{D}$: The YOY method for finding degradation trends in PV production data has been well documented previously [2], [5]–[7]. In the CODS algorithm, the input to the YOY method is $\frac{PI}{SR \cdot SC}$. Although the YOY method is insensitive to seasonal trends, the YOY method benefits from removal of soiling component from the input signal, as was shown in [8]. The YOY method works in spite of missing data, but it needs a minimum of two years of data. This puts a lower bound on the amount of data needed to run the CODS algorithm.

5) Putting It All Together: The CODS algorithm will run until it converges, or until it reaches the maximum number of iterations set by the user. In this context, convergence is measured by the root-mean-squared error (RMSE) of the daily values of the model fit versus the observed *PI*. The convergence criterion

is that the relative change in the RMSE from one iteration to the next is smaller than 0.5%. However, the model is allowed to converge twice. Initially, perfect cleaning is assumed during the estimation of SR, as described in Section II-B.2. When the algorithm first converges, it is assumed that we have found sufficiently accurate estimates of SC and D; The seasonality- and degradation-corrected input signal to the KF is then assumed to only contain the soiling signal, and the assumption of perfect cleaning is dropped. The algorithm is then allowed to iterate further until it converges a second time.

In some cases, two different phenomena may have the same signature on the *PI* time series. For instance, in locations with rain- and dry-seasons, there might be seasonal patterns in the soiling signal causing the *PI* to be lower in the dry season than in the rain season. In some cases, it will be challenging to differentiate between a seasonal pattern in the soiling and a seasonal component. The order in which the three components of the model are obtained often decides which component a certain data signature is ascribed to. Although the problem of correctly assigning features to the right component is mostly relevant for *SR* and *SC,* it is possible to change the order in which all three components are found, or to omit one of the components entirely. This is indicated by the dashed arrows in Fig. 1. Each combination of component estimation orders constitutes a separate model choice. In the bootstrapping procedure, described in Section II-C.2, the algorithm is run alternating between finding SR before SC, and vice versa, ensuring that the uncertainty associated with this effect is accounted for. The choice of component estimation order and the fill-method for missing data during cleaning event detection described in Section II-B.1 makes up the possible CODS model choices.

Since the main input to the expected energy term in the *PI* is often the daily insolation, the *PI* is especially sensitive to systematic errors in this value. Two likely causes of systematic errors are soiling of the irradiance sensor, which will mostly bias the *SR* estimate, and sensor drifting, which will mostly bias the D estimate. Comparing analysis made on *PI*-data based on irradiance data from on-site sensor measurements, satellite measurements, and a clear-sky model may help mitigate this source of error [5], [21]. Another prominent source of error is grid or inverter downtime. It is advisable to remove such datapoints before running the model. Another possible source of systematic error is correlation between soiling and temperature. Ideally, this should be accounted for in the modeling of the expected energy when calculating *PI*. Finally, it is advisable to minimize the noise in the *PI.* One way to do this is to filter out cloudy conditions, low solar elevation angles, and high angles of incidence before aggregating to daily values.

If the soiling signal is smaller than the noise, the algorithm fails to estimate SR accurately. To avoid this problem, the CODS algorithm checks the relative magnitude of the soiling signal and the noise. It does this by comparing the 95-percentile span of the estimated SR and n as follows:

$$\frac{P97.5\,(SR)\ -\ P2.5\,(SR)}{P97.5\,(n)\ -\ P2.5\,(n)} < 0.75 \tag{2}$$

where $Px(S)$ is the xth percentile of S. If condition (2) is true, the dataset is flagged as having an insignificant soiling signal, and the CODS algorithm is not run. Instead, only the YOY method is run, estimating the degradation rate R_d.

To assess whether the fitting procedure has been successful or not, we check two conditions: Whether the soiling signal is significant, and whether the residuals are *stationary,* i.e., whether the probability distribution describing the residuals changes through time. If the residuals are not stationary, it means there is information in the residuals that has not been captured by the fitting procedure, and the fit is considered unsuccessful. Whether the residuals are stationary or not is tested using an *augmented Dickey–Fuller test*, which is a statistical test for stationarity in time series [22]. We use the statsmodels implementation of the test, checking for both constant, linear and quadratic trends at a significance level of 5%.

Depending on the length of the time series, a single run of the CODS algorithm takes between a few seconds and half a minute.

C. Estimating Uncertainty With Bootstrapping

Bootstrapping is the process of randomly sampling with replacement from a pool of data, thus generating new, independent datasets. In our case, we employ bootstrapping to estimate the uncertainty of our model fit. We do this in three stages—in stage 1, we fit a set of CODS models to the original *PI* data and generate a set of bootstrap samples based on these fits. In stage 2, we fit a set of CODS models to the bootstrap samples. In both stages, we vary the model and parameter choices of each fit. In this way, uncertainty in the data itself, as well as uncertainty associated with the choice of model and parameters, is accounted for. In stage 3, confidence intervals and final component estimates are found based on the model fits of stages 1 and 2.

1) Stage 1. Generating Bootstrap Samples: To generate bootstrap samples from time series data, we use *circular block bootstrapping*. The original bootstrap idea requires independently and identically distributed samples, which would exclude the kind of time series we are considering. However, later variants of bootstrapping, such as sieve bootstrapping [23] and moving block bootstrapping [24] allow autocorrelated samples to be bootstrapped, as long as the residuals are stationary. In this work, we follow the ideas proposed in [25] and [26] to generate bootstrap samples. Although these studies are about time series forecasting, the bootstrapping part is relevant for our application. The bootstrapping approach as implemented for CODS can be seen in Fig. 2, and is described in the following.

The procedure starts by fitting a CODS model to the data. Next, bootstrap samples are generated from the model residuals using circular block bootstrapping. This involves sampling random 90-day blocks of residuals with replacement, appending them to each other until the resampled time series of residuals has the same length as the original time series. The term *circular* refers to the time series being "wrapped in a circle" before resampling. In other words, sample blocks are allowed to start close to the end of the time series, "spilling over" into the beginning of the time series. Doing this, we avoid undersampling data points close to the edges of the time series. Bootstrapped *PI*-signals are then generated by the product of the model fit and

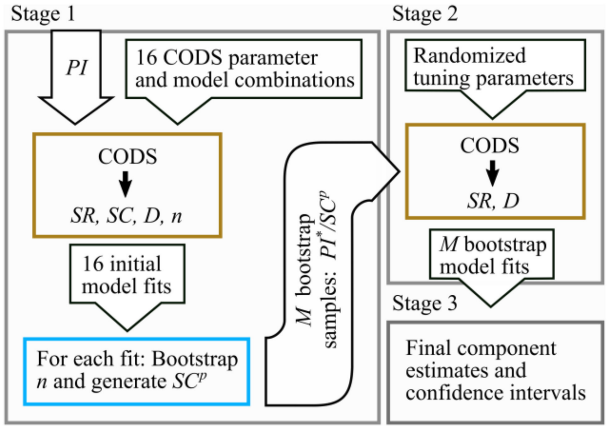

Fig. 2. Flowchart for the bootstrapping procedure for estimating the uncertainty of the components found by the CODS algorithm. In stage 1, 16 models with different tuning parameters are fit with the CODS algorithm. Then, M bootstrap samples are generated by doing circular block bootstrapping of the residuals, multiplying the bootstrapped residuals by the model fits. In stage 2, the CODS algorithm is run on each of the M bootstrap samples, fitting M models. From the results of the M fits, confidence intervals, and final component estimates are found in stage 3.

TABLE I
DEFAULT PARAMETER AND MODEL CHOICES OF THE CODS MODEL

Parameter	**Low**	**High**
Cleaning event detection sensitivity tuner, *a*	0.4	0.8
Cleaning event pruning sensitivity tuner, *b*	0.75	1.25
Model choice	**Alt. 1**	**Alt. 2**
Fill-method for missing data in cleaning event detection	Forward fill	Backward fill
Component estimation order	*SR, SC, D*	*SC, SR, D*

the bootstrapped residuals, creating a time series similar to the original *PI,* but with different noise. We will denote these signals by PI^*.

The problem with the above approach, if this was all that happened in stage 1, is that any bias in the initially fitted model will carry over into the generated PI^*. The total uncertainty can be viewed as a combination of *parameter* and *model* uncertainty, i.e., uncertainty associated with the choice of parameters and model, respectively [26]. To account for this uncertainty, instead of generating the bootstrap samples based on a single initial fit, we make fits with each possible combination of the two tuning parameters (cleaning event detection and pruning sensitivity) and model choices (fill-method for cleaning event detection and component estimation order) summarized in Table I, for a total of 16 fits.

We then apply the approach for generating PI^* signals described above to each of the initial 16 model fits, thus generating a predefined number (*M*) of bootstrap samples. The default values for the tuning parameters in Table I have been determined by manual tuning based on the synthetic data described in Section II-D. In a dataset from the field, a different set of tuning parameter values might be needed. We recommend that these are adjusted by visually validating the cleaning detection sensitivity and pruning sensitivity. For large scale applications on many systems, we similarly recommend spot checking results.

2) Stage 2. Fitting the Model to the Boostrap Samples: Once the *M* bootstrap signals have been generated, a condensed version of the CODS algorithm is applied to each of the PI^* signals. In this stage, the seasonal component is not estimated. Judging from our experience, SC is the most difficult component to estimate accurately, especially when it is small relative to SR. However, SC is included in the model to help estimate SR more accurately, and we are not really interested in its exact value. Since there is a high uncertainty associated with SC, we need a heuristic for introducing this uncertainty in the bootstrapping procedure. We have found that the best way of doing this is to generate a set of perturbed seasonal components based on the 16 initial model fits. This is done as follows: Let $\widetilde{SC}$ be the weighted average of all the estimated SCs of the 16 initial model fits. The weights are defined by

$$w = \frac{1}{e\,(1+m)} \tag{3}$$

where e is the RMSE of the daily values of the model fit and the *PI,* and m is the fraction of time where SR has been capped (because it was estimated to be above 1, see Section II-B.2). If m is high, it means the *SR* has been overestimated in much of the time series. Thus, by weighting by w, fits with a low error and a low fraction of overestimated *SR* are favored. For each of the *M* bootstrap samples, we generate a perturbed seasonal component SC^p by multiplying the amplitude of $\widetilde{SC}$ by a random factor between 0.8 and 1.75, and shifting its phase by a random number of days between -30 and 30. These numbers have been decided by trial and error on the synthetic data described in Section II-D, and they ensure that the 95% confidence intervals are consistent. We then run the CODS algorithm on $\frac{PI^*}{SC^p}$. We now assume the seasonal component has been removed from the signal, and only the soiling and degradation components are estimated, in that order.

To further account for parameter uncertainty, in this stage we run the model with random tuning parameters and a random choice of forward filling and backward filling during cleaning event detection. The tuning parameters are chosen from uniform distributions. The bounds of these distributions are the parameter values summarized in Table I.

Furthermore, at this stage there are two additional input parameters that are varied: the *process noise* and the *SR renormalization factor*. The process noise is an input parameter to the Kalman filter that, simply put, determines how quickly the filter adapts to changes in the input signal. We initialize the process noise matrix through the built-in function from FilterPy called Q_discreet_white_noise. The input to this function is randomly sampled from a uniform distribution between $6.67e^{-5}$ and $1.5e^{-4}$. These bounds have been determined by manual tuning based on the synthetic data described in Section II-D. The *SR* renormalization factor, on the other hand, adjusts the absolute value of the *SR*. It has a 50/50 chance of being ON or OFF. When it is on, *SR* is normalized by the *k*th percentile of the set of *SR* values on days just after cleaning events. *k* is determined by random sampling from a uniform distribution between 5 and

95. This accounts for the uncertainty in the absolute value of the *SR,* associated with how effective the cleaning has been.

3) Stage 3. Final Component Estimates and Confidence Intervals: The confidence intervals for the daily values of SR and R_{SR} are estimated from the distributions of the bootstrapped fits of the respective components. The final estimates for SR and R_{SR} are given by the best of the initial 16 model fits, determined by the maximum *w,* defined in (3). When estimating the average soiling loss over the whole period, we use the weighted average of the result in each of the successful bootstrap samples, where the weights are *w.*

The final estimate of the degradation rate is found in the same way as the average soiling loss. The confidence intervals are found from the percentile distribution of the set of bootstrap estimates of the degradation rate. We note that this is a different way of estimating confidence intervals than what is done in the RdTools (version 1.x.x) implementation of the YOY method, degradation_year_on_year. There, the confidence intervals are constructed from a bootstrapping of the set of YOY rates in the data. In the CODS approach, the influence of both data uncertainty and the uncertainty in the soiling signal and the seasonal component on the degradation rate is accounted for. In general, the mutual influence on uncertainty of all three components is accounted for by our bootstrapping procedure. This is not the case in the previous RdTools implementation of the YOY method.

Even if the user is not interested in confidence intervals, we recommend running the CODS algorithm with bootstrapping, as this both gives better estimates of the components and inherently takes care of parameter and model choices. We suggest using 512 bootstrap samples, i.e., $M = 512$. For a 10-year dataset, this takes approximately 20 min on a normal laptop from 2018, and the computation times scales approximately linearly with the length of the time series. This means the algorithm is applicable to both fleet-scale level analysis and inverter level analysis in utility scale power plants.

D. Generation of Synthetic Data

In order to validate the model, a framework for generating synthetic *PI* time series was made.[2] The advantage of synthetic data, in contrast with data from the field, is that we can know the exact value of the separate components, which will allow us to assess the accuracy of the model on each component. We have generated the data following the same framework as in [8]: SR is generated by randomly selecting 120 cleaning events over a ten year period, and letting SR reduce linearly from 1 at a soiling rate randomly sampled from a uniform distribution between $\mathrm{r_{s,min}}$ and $\mathrm{r_{s,max}}$ for each section between two cleaning events. SC is generated as a sine wave centered at 1, with an amplitude A_{seas} and a period of one year. D is generated as a linear trend starting at 1 and decreasing at a rate of $R_d = -0.5\%/\mathrm{year}$. n is generated as Gaussian white noise centered at 1 and with standard deviation σ_{noise}.

[1]The Python code used to generate these time series is available at https://github.com/asmunds/simulate_pv_time_series/tree/021d5724806e93829dedee376a149199a068ed79

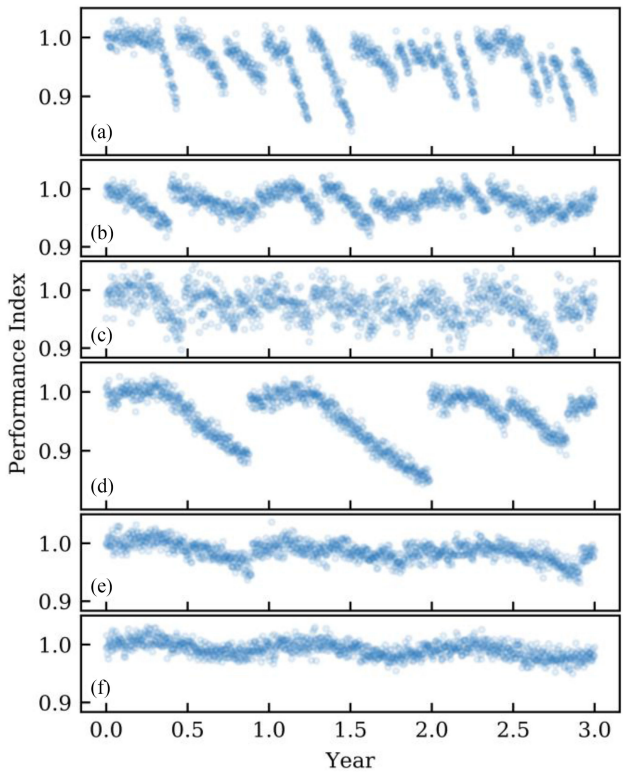

Fig. 3. Realizations of the six different versions of the generated synthetic daily *PI*. The parameters of each instance are summarized in Table II.

TABLE II
GENERATION PARAMETERS OF SYNTHETIC DATA SCENARIOS

Instance	R_d (%/year)	A_{seas} (%)	σ_{noise} (%)	$r_{s,min}$ (%/day)	$r_{s,max}$ (%/day)
a)	-0.5	1	1	0	0.3
b)	-0.5	2	1	0	0.1
c)	-0.5	1	2	0	0.1
d)*	-0.5	1	1	0.05	0.05
e)	-0.5	1	1	0	0.05
f)	-0.5	1	1	0	0.01

*Generated with a seasonal variation in the cleaning frequency

In the field, *PI* time series will have a wide range of different soiling intensities, seasonal effects in the soiling, other seasonal effects (e.g., incident angle, shading, and temperature effects), and noise levels. Since we do not have access to fielded data that represents the variation we expect, we have come up with six different synthetic data scenarios that we believe represents this variety. One realization of each of these models is shown in Fig. 3, and the parameter combinations are summarized in Table II. The hope is that these six parameter combinations span much of the variety of *PI* time series from the field, and that by demonstrating the accuracy of the CODS model on these datasets, we convince the reader that it will be accurate on data from the field as well.

Note that instance d) has a seasonal variation in the cleaning frequency. To achieve this, a probability distribution representing the probability of cleaning for each day d in the time series

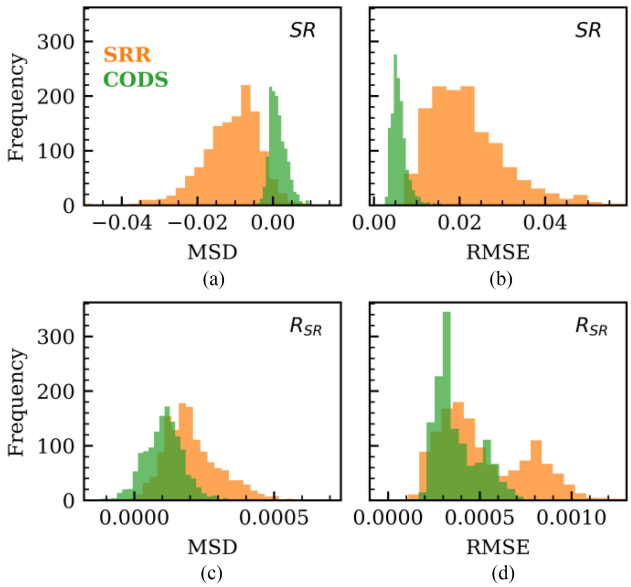

Fig. 4. Error metrics for estimated daily values of SR and R_{SR} based on 2376 realizations of the CODS algorithm. (a) Distribution of MSD of the estimate of SR versus the truth for the realizations. (b) RMSE for the same. (c) MSD for the estimate of R_{SR} versus the truth. (d) RMSE for the same.

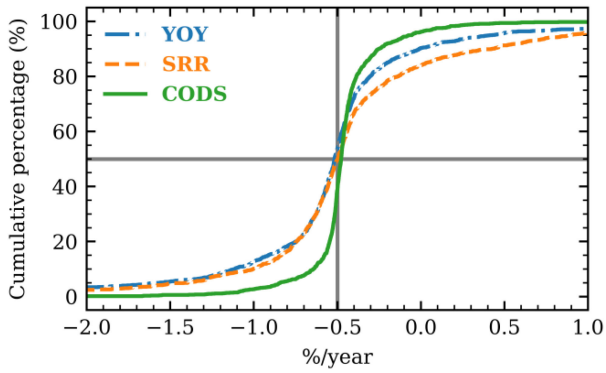

Fig. 5. Cumulative percentage of estimated degradation rates based on 2376 realizations of the synthetic data. The true degradation rate is -0.5%/year in all realizations, illustrated by the vertical grey line. The CODS algorithm generally estimates degradation rates closer to the truth than the other two methods.

is constructed as:

$$P\ (d) = A\left(1.1 + \sin\left(\frac{d}{365.25} + x\right)\right). \tag{4}$$

Here, x represents a random phase, and A is set so that the area under the curve integrates to one. The cleaning events are sampled, without replacement, from the set of days in the time series with the sampling probability given by this probability distribution. This emulates environments with periodic variation in the probability of rain (such as locations with rainy seasons).

III. Results and Discussion

A. Validation on Synthetic Data

The model was tested on 2376 realizations of the synthetic data described in Section II-D, with an equal number of each of the 6 variations shown in Fig. 3, and an equal number of two-year, five-year, and ten-year datasets. Using these synthetic data, we were able to confirm the validity of the confidence intervals of the CODS algorithm. In the following, the accuracy of the estimates of the soiling ratio (SR), soiling rates (R_{SR}), and degradation rates (R_d) are assessed.

1) Soiling Ratio and Soiling Rates: The accuracy of the CODS algorithm's estimate of daily values of SR and R_{SR}, measured in mean signed deviation (MSD) and RMSE, is shown by histograms in Fig. 4. The results are compared with estimates made with the stochastic rate and recovery (SRR) method, which is described in [3]. The MSD of the SR estimates show that the CODS algorithm is clearly less biased than the SRR method. Further, the RMSE results show that it in general is more accurate. The same results hold if we look at monthly or yearly averages, instead of daily values.

When it comes to R_{SR}, the improvement of the CODS algorithm versus the SRR method is smaller. Although CODS seems to be slightly less biased, judging from the MSD histogram, and slightly more accurate, judging from the RMSE histogram, the improvement is not large. The bias comes from the difficulty of estimating soiling rates in periods with quick successive cleaning events, since the soiling signal remains smaller than the noise in these periods, leading to underestimated soiling rates and a positive MSD. This issue is not specific to the CODS model; it is a general issue for all models attempting to estimate soiling rates based on a daily *PI*.

2) Degradation Rates: Based on the 2376 realizations, the degradation rates found with the YOY method alone, the SRR method combined with the YOY method (denoted SRR in Fig. 5), and the CODS algorithm, are shown in Fig. 5. The combined use of SRR and YOY was done according to [8]. In all cases, the true degradation rate is $-0.5\%/$year, which is shown as a vertical line in Fig. 5. The cumulative distribution of estimated degradation rates shows that, based on the synthetic data used in this article, the combined use of SRR and YOY does not constitute an improvement relative to using the YOY method alone. This is in contrast with what was found in [8]. The reason for this, we believe, is that the synthetic data used in [8] was consistently generated in the same way as dataset a) in Table I, which has a very large soiling signal relative to the other signals. When the soiling signal is large, it is indeed useful to correct for soiling using the SRR method, before running the YOY method. We were able to reproduce these results when using only dataset a). However, when the soiling signal is small, when there is a high amount of seasonality, or when the amplitude of the noise is large, using the SRR method to correct for soiling does not necessarily lead to a more accurate estimate of the degradation rate.

On the other hand, the degradation rates found with the CODS algorithm are more consistently closer to the true degradation rate than those found by the YOY method alone. Although this is not shown in Fig. 5, this holds even in datasets where the soiling signal is small. This is as expected, because accurately estimating SR and correcting for it in the input to YOY, which is effectively what is happening in the CODS algorithm, will remove bias caused by soiling in the YOY estimate.

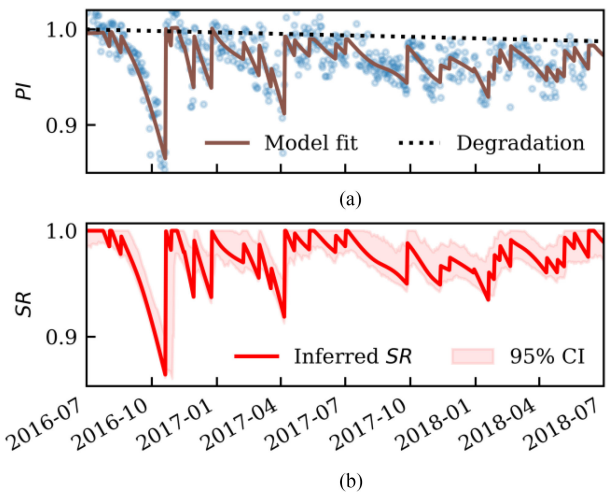

Fig. 6. CODS model applied to field-data. (a) Daily PI along with the model fit and the estimated degradation trend (-0.6%/year). (b) Inferred SR along with its 95% confidence interval. The estimated average soiling loss is 2.2%.

B. Validation on Field Data

Fig. 6 shows daily *PI* for a fielded PV system, along with the associated CODS results. The *PI* used here is the temperature corrected performance ratio [10] of the dc power of an inverter in a utility scale PV power plant.

Since we do not know the true values of daily soiling losses and degradation rates, we can only make a visual assessment of how well the model fits the data. Looking at Fig. 6, the model seems to fit the data well. Judging from the coefficient of determination, $R^2 = 0.71$, the model accounts for 71% of the variation in the data. This shows that, even though the choices made in the model development, as well as the chosen set of model parameters, have been made based on synthetic data, the model works on at least one example of field data.

In the PV system in study, the estimated average soiling loss is 2.2%, with a 95% confidence interval (CI) between 1.5% and 3.2%. The estimated degradation rate is -0.6%/year, with a 95% CI between 0.0%/year and -1.1%/year. The large confidence interval reflects the large amount of noise, and the large soiling signal. In contrast, applying the RdTools implementation of the YOY approach with no soiling correction yields a degradation rate of -0.2%/year, with a 95% CI between $+0.1\%$/year and -0.6%/year. The estimated average soiling rate is $R_{SR} = -0.1\%$/day.

IV. Conclusion

We have presented an algorithm for simultaneous estimation of soiling losses and degradation rates in PV systems based on time series of a daily PI. Based on 2376 realizations of synthetic data, we have shown that it outperforms related state-of-the-art models in terms of its accuracy in estimating degradation rates and daily soiling losses, as well as monthly and yearly averages. Even in moderate soiling conditions, the model gives more accurate estimates of the degradation rates than the YOY method. It takes about 20 min to complete a full analysis of a 10-year dataset on a standard computer from 2018.

A preliminary version of the CODS model is freely available for anyone to use, and will further be distributed through a future release of RdTools.

Acknowledgment

The authors would like to thank T. Silverman, K. Perry, H. Haug, and E. S. Marstein for helpful discussions during the development of the CODS-method.

The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this article, or allow others to do so, for U.S. Government purposes.

References

[1] K. Ilse *et al.*, "Techno-economic assessment of soiling losses and mitigation strategies for solar power generation," *Joule*, vol. 3, pp. 2303–2321, 2019.

[2] D. C. Jordan and S. R. Kurtz, "Photovoltaic degradation rates - An analytical review," *Prog. Photovolt. Res. Appl.*, vol. 21, no. 1, pp. 12–29, 2013.

[3] M. G. Deceglie, L. Micheli, and M. Muller, "Quantifying soiling loss directly from PV yield," *IEEE J. Photovolt.*, vol. 8, no. 2, pp. 547–551, Mar. 2018.

[4] Å. Skomedal, H. Haug, and E. S. Marstein, "Endogenous soiling rate determination and detection of cleaning events in utility-scale PV plants," *IEEE J. Photovolt.*, vol. 9, no. 3, pp. 858–863, May 2019.

[5] D. C. Jordan, C. Deline, S. R. Kurtz, G. M. Kimball, and M. Anderson, "Robust PV degradation methodology and application," *IEEE J. Photovolt.*, vol. 8, no. 2, pp. 525–531, Mar. 2018.

[6] E. Hasselbrink *et al.*, "Validation of the PVLife model using 3 million module-years of live site data," in *Proc. Conf. Rec. IEEE Photovolt. Specialists Conf.*, 2013, pp. 7–12.

[7] D. C. Jordan, M. G. Deceglie, and S. R. Kurtz, "PV degradation methodology comparison - A basis for a standard," in *Proc. IEEE 44th Photovolt. Specialists Conf.* , 2017, pp. 1–6.

[8] M. G. Deceglie, M. Muller, D. C. Jordan, and C. Deline, "Numerical validation of an algorithm for combined soiling and degradation analysis of photovoltaic systems," in *Proc. IEEE 46th Photovolt. Specialists Conf.*, 2019, pp. 3111–3114.

[9] NREL, "RdTools, version 1.2.2," 2019. [Online]. Available: https://github.com/NREL/rdtools

[10] IEA, *Photovoltaic System Performance - Part 1: Monitoring*, IEC 61724-1, IEC, Mar. 2017.

[11] A. Kimber, L. Mitchell, S. Nogradi, and H. Wenger, "The effect of soiling on large grid-connected photovoltaic systems in California and the Southwest Region of the United States," in *Proc. Conf. Rec. IEEE 4th World Conf. Photovolt. Energy Convers.*, 2007, pp. 2391–2395.

[12] J. R. Caron and B. Littmann, "Direct monitoring of energy lost due to soiling on first solar modules in California," *IEEE J. Photovolt.*, vol. 3, no. 1, pp. 336–340, Jan. 2013.

[13] L. Micheli, E. F. Fernandez, M. Muller, and F. Almonacid, "Extracting and generating PV soiling profiles for analysis, forecasting, and cleaning optimization," *IEEE J. Photovolt.*, vol. 10, no. 1, pp. 197–205, Jan. 2020.

[14] R. Cleveland, W. Cleveland, J. McRae, and I. Terpenning, "STL: A seasonal-trend decomposition procedure based on Loess (with discussion)," *J. Official Statist.*, vol. 6, no. 1. pp. 3–73, 1990.

[15] R. R. Labbe, "Kalman and Bayesian Filters in Python," 2016. [Online]. Available: https://elec3004.uqcloud.net/2015/tutes/Kalman_and_Bayesian_Filters_in_Python.pdf

[16] H. Musoff and P. Zarchan, *Fundamentals of Kalman Filtering: A Practical Approach*, 3rd ed. Reston, VA, USA: Amer. Inst. Aeronautics Astronautics, 2009.

[17] R. R. Labbe, "FilterPy, version 1.4.5," 2015. [Online]. Available: https://github.com/rlabbe/filterpy
[18] H. E. Rauch, F. Tung, and C. T. Striebel, "Maximum likelihood estimates of linear dynamic systems," *AIAA J.*, vol. 3, no. 8, pp. 1445–1450, 1965.
[19] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with Python," in *Proc. 9th Python Science Conf.*, 2010, vol. 57, pp. 92–96.
[20] S. Seabold and J. Perktold, "Statsmodels version 0.11.0," 2020. [Online]. Available: https://www.statsmodels.org/stable/
[21] Z. Defreitas *et al.*, "Evaluating the accuracy of various irradiance models in detecting soiling of irradiance sensors," Preprint. Golden, CO: National Renewable Energy Laboratory. NREL/CP-5K00-75156. [Online]. Available: https://www.nrel.gov/docs/fy20osti/75156.pdf
[22] S. E. Said and D. A. Dickey, "Testing for unit roots in autoregressive-moving average models of unknown order," *Biometrika*, vol. 71, no. 3, pp. 599–607, 1984.
[23] P. Bühlmann, "Sieve bootstrap for time series," *Bernoulli*, vol. 3, no. 2, pp. 123–148, 1997.
[24] H. R. Kunsch, "The jackknife and the bootstrap for general stationary observations," *Ann. Statist.*, vol. 17, no. 3, pp. 1217–1241, 1989.
[25] C. Bergmeir, R. J. Hyndman, and J. M. Benítez, "Bagging exponential smoothing methods using STL decomposition and Box–Cox transformation," *Int. J. Forecasting*, vol. 32, no. 2, pp. 303–312, 2016.
[26] F. Petropoulos, R. J. Hyndman, and C. Bergmeir, "Exploring the sources of uncertainty: Why does bagging for time series forecasting work?," *Eur. J. Oper. Res.*, vol. 268, no. 2, pp. 545–554, 2018.

Åsmund Skomedal was born in Søgne, Norway, in 1991. He received the M.S. degree in applied physics from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 2017 and is currently working toward the Ph.D. degree in PV systems with the Institute for Energy Technology (IFE), Kjeller, Norway, collaborating with the University of Oslo (UiO), Oslo, Norway.

As a part of the Ph.D. program, he was lucky to be on a three month visit to the National Renewable Energy Laboratory (NREL) in Golden, CO, USA, in the autumn of 2019, where he worked with Michael G. Deceglie developing the model presented in this article.

Michael G. Deceglie received the B.S. degree in physics from Dickinson College, Carlisle, PA, USA, in 2006, the M.Phil. degree in physics from the University of Queensland, Brisbane, Australia, as a Fulbright Fellow, and the Ph.D. degree in applied physics from Caltech, Pasadena, CA, USA, in 2013.

He is currently a Staff Scientist with the National Renewable Energy Laboratory, Golden, CO, USA, studying PV field performance and reliability.