Forecasting volatility with a stacked model based on a hybridized Artificial Neural Network

DHENIN Alexandre - LEFEVRE Cassandre - MANELLI Nicolas - NITCHEU Rayan

1 Introduction

La prévision de la volatilité a longtemps été un aspect important de la gestion des risques financiers et de la prise de décision en matière d'investissement. Prédire avec précision la volatilité future des actifs peut fournir aux investisseurs des informations précieuses pour gérer leurs portefeuilles et prendre des décisions d'investissement éclairées.

D'autre part, ces dernières années, grâce à l'augementation de la puissance des calculs, l'apprentissage automatique s'est imposé comme un outil puissant de modélisation financière et a montré des résultats prometteurs en matière de prédictions. L'objectif de cette présentation est d'expliquer le papier "Forecasting volatility with a stacked model based on a hybridized Artificial Neural Network" est de présenter nos implémentations. Nous fournirons une vue d'ensemble des concepts et méthodologies clés, ainsi qu'une évaluation critique des forces et des limites de chaque approche. De plus nous présenterons en détail le modèle d'Hestoncen guise de Benchmark.

2 Benchmark

2.1 ANN

Un réseau de neurone est un algorithme d'apprentissage automatique basé sur le modèle d'un neurone humain, c'est est une technique de traitement de l'information. Il fonctionne de la même manière que le cerveau humain traite les informations. Ils sont capables d'apprentissage automatique ainsi que de reconnaissance de formes. Ils se présentent comme des systèmes de "neurones" interconnectés qui peuvent calculer des valeurs à partir d'entrées. Un réseau de neurones contient les 3 couches suivantes :

- Couche d'entrée : l'information brute alimentant le réseau.
- Couche cachée : Les activités des unités d'entrée et les poids des connexions entre l'entrée et les unités cachées. Il peut y avoir une ou plusieurs couches cachées.
- Couche de sortie : Le comportement des unités de sortie dépend de l'activité des unités cachées et des poids entre les unités cachées et de sortie.

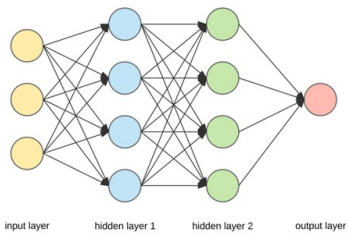

Figure 1: Illustration de Réseau de Neurones.

2.2 GARCH et GJR-GARCH

Le modèle GARCH (Generalized AutoRegressive Conditional Heteroskedasticity) est un modèle statistique utilisé dans l'analyse des données de séries temporelles. Les modèles GARCH supposent que la variance du terme d'erreur suit un processus de moyenne mobile autorégressive. Les institutions financières les utilisent généralement pour estimer la volatilité des rendements des actions, des obligations et des indices boursiers. Elles utilisent les informations qui en résultent pour déterminer les prix et juger quels actifs sont susceptibles de fournir des rendements plus élevés, ainsi que pour prévoir les rendements des investissements actuels afin de les aider dans leurs décisions d'allocation d'actifs, de couverture, de gestion des risques et d'optimisation de portefeuille.

$$\sigma_t^2 = \omega + \sum_{p=1}^{P} \alpha_p \epsilon_{t-p}^2 + \sum_{q=1}^{Q} \beta_q \sigma_{t-q}^2 \tag{1}$$

Le modèle GJR-GARCH est une autre forme du modèle GARCH. Le modèle GJR-GARCH a été mis en place pour surmonter la faiblesse du traitement GARCH des séries temporelles financières. En particulier, pour permettre des effets asymétriques. Le modèle admet l'ajout d'un effet supplémentaire lorsque le rendements précédent est négatif.

$$\sigma_t^2 = \omega + \sum_{p=1}^{P} \alpha_p \epsilon_{t-p}^2 + \sum_{q=1}^{Q} \beta_q \sigma_{t-q}^2 + \gamma_p \epsilon_{t-p}^2 \mathbf{1}_{\{\epsilon_{t-p}<0\}} \tag{2}$$

2.3 ANN - GARCH et ANN – EGARCH

Les réseaux de neurones superforme les autres modèles. Cependant les prédictions de volatilités deviennent imprécises si les prix évoluent fortement. Alors il est nécessaire de combiner un réseau de neurones avec un autre modèle. En effet, l'idée est d'entrée en input de notre réseau de neurones, les volatilités prédites obtenue en output d'un autre modèle comme un GARCH. On nommera ce modèle un « ANN-GARCH ». Ainsi nous avons implémenté un modèle GARCH et EGARCH.

2.4 Modèle de Heston

Le modèle de Heston, développé par Steve Heston, est un modèle de volatilité fréquemment utilisé pour évaluer la valeur des options exotiques. Il repose sur l'hypothèse que la volatilité de l'actif est un processus stochastique appartenant à la famille des processus de Cox Ingersoll Ross qui présentent un comportement de "mean-reverting". Ces processus peuvent reproduire des phénomènes tels que les "Fat Tails", les "High Peaks" et le "Skewness" dans la volatilité implicite, qui ne sont pas couverts par les modèles à volatilité constante.

$$\begin{aligned} dS_t &= \mu S_t \, dt + \sqrt{V_t} S_t dB_t^S \\ dV_t &= \kappa(\theta - V_t) \, dt + \sigma \sqrt{V_t} dB_t^v \end{aligned} \tag{3}$$

Le modèle de Heston comporte plusieurs paramètres qui doivent être estimés pour une utilisation pratique. Les paramètres les plus importants incluent la volatilité long terme θ, la vitesse de retour à la moyenne κ, la corrélation entre la volatilité et la performance de l'actif sous-jacent ρ, le paramètre de diffusion pour la volatilité σ, et finalement un paramètre apparaissant sous la dynamique risque neutre λ representant un premium lié au risque induit par le caractère stochastique de la volatilité. La détermination de ces paramètres peut se faire par l'utilisation de différentes méthodes de calibrations, notamment via la minimisation d'une fonction de coût a l'aide d'un algorithme d'optimisation (dans notre cas celui de The Broyden, Fletcher, Goldfarb, and Shanno). Il est nécessaire de procéder a une calibration régulière du modèle en raison de la nature dynamique de ses paramètres. De même, il est important de noter que la qualité des résultats dépend fortement de celle de la calibration des dits paramètres. Le modèle de Heston est principalement utilisé pour générer une surface de volatilité implicite conforme à celle du marché. Cependant, dans ce projet, il sera utilisé comme référence pour

comparer le Stacked-ANN avec ses alternatives.

Nous avons rencontré des difficultés pour reproduire les résultats du papier original. Premièrement, le modèle d'Heston nécessite de disposer des surfaces de volatilité de manière journalière afin de se calibrer, or, ces données ne sont généralement pas accessible au grand public. De plus, aucune méthodologie précise n'a été évoquée pour effectuer le test du Heston. Nous avons donc tenté de suivre un protocole similaire a celui utilisé pour entraîner la première couche de modèle a partir des surfaces de vol s'étendant sur la période 2009 - 2016 :

- 75% des données ont été utilisées afin de calibrer le modèles et de trouver les différents paramètre de notre dynamique de volatilité.
- Prédiction de la volatilité réalisée sur les période restante.

Concernant les valeurs obtenues en résultats, nos paramètres sont les suivants :

	prédite	usuelles
variance	0.039345	0.0625
kappa	0.127164	2
theta	0.055037	0.04
sigma	0.001981	0.6
rho	-0.758754	-0.7
lambda	-0.07028	-0.5

Figure 2: Résultats obtenus pour nos paramètres contre les valeurs usuelles.

Les écarts de valeur entre nos paramètres prédits et usuels peuvent notamment être expliqués par la nature de notre protocole. En effet, comme sus-mentionné, le Heston est un modèle paramétrique qui nécessite d'être calibrer de manière dynamique afin de s'adapter aux conditions de marché changeantes.

3 Modèle de Stacked-ANN

Le Stacked-ANN est un modèle hybride combinant plusieurs modèles comme le ANN-GARCH. En effet, l'idée est d'entrée en input de notre Réseau de Neurones, les volatilités prédites obtenue en output d'un modèle Random Forest, SVM et Gradient Boosting.

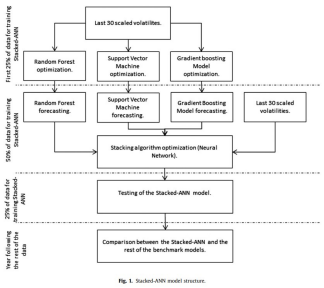

Figure 3: Illustration de l'architecture Stacked-ANN.

3.1 Test de Dickey-Fuller Augmenté

Le test de ADF permet de vérifié la non stationnarité d'une série temporelle. La statistique de test est comparée aux valeurs critiques de la distribution. Si la statistique de test est inférieure aux

valeurs critiques, cela suggère que les données sont stationnaires. Dans notre cas, on observe une série stationnaire sur toute les période sauf sur 2009 - 2016 4.

	Test Statistic	p-value	#Lags Used	Number of Observations Used	Critical Value (1%)	Critical Value (5%)
2000:2007	-4.567741	0.000148	11	423	-3.445904	-2.868397
2000:2008	-4.872523	0.000039	11	479	-3.444076	-2.867593
2000:2009	-5.020157	0.000020	9	536	-3.442609	-2.866947
2009:2016	-2.429764	0.133511	6	435	-3.445473	-2.868207
2010:2017	-3.030622	0.032132	12	428	-3.445721	-2.868317

Figure 4: ADF test.

3.2 Random Forest

Le Random Forest [2] est un algorithme d'apprentissage supervisé qui combine plusieurs arbres de décision ou de régression dans l'optique de créer une "forêt", soit une multitude d'arbre. Un arbre de régression est une procédure classifiant dans des régions un set de donnée, puis qui prédit la moyenne de chaque régions de la variable cible. La procédure du Random Forest étant une technique de bagging, elle a pour but de généralisé un problème, d'en réduire la variance et d'améliorer les prédictions. Elle possède un aspect aléatoire étant le choix de features, qui permet de créer des arbres non corrélé. Une fois les arbres construit, la moyenne de ces derniers est fait afin de construire le model final.

$$F(x) = \frac{1}{n}\sum_{i=1}^{n} F_i(x) \tag{4}$$

where n is the number of trees and F_i is the ith tree in the forest.

Algorithm 1 Random Forest Pseudocode

procedure RANDOMFOREST($X, y, nb^{trees}, m^{Try}$)
  Initialize an empty list to store the trees
  for $i = 1$ to nb^{trees} **do**
    (1) Draw a random sample of m features with replacement from the total set of features
    (2) Fit a regression tree to the sub-sample of features and the corresponding target values
    (3) Add the fitted tree, $F_i(x)$, to the list of trees
  end for
  return $F(x) = \frac{1}{nb^{trees}} \sum_{i=1}^{nb^{trees}} F_i(x)$
end procedure

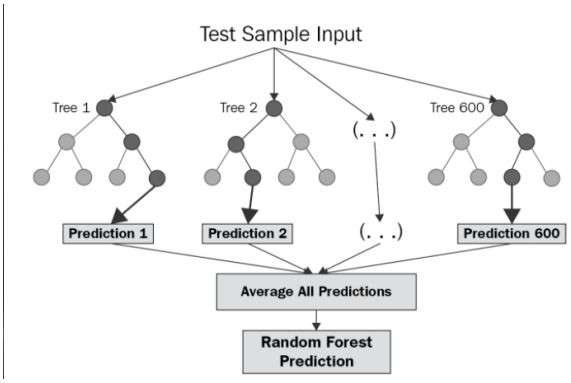

Figure 5: Illustration du modèle de Random Forest.

3.3 Gradient Boosting

Le Gradient Boosting [2] est une technique d'apprentissage pour les problèmes de régression et de classification qui produit un modèle de prédiction sous la forme d'un ensemble de modèles de prédiction faibles. Cette technique construit un modèle par étapes et le généralise en permettant l'optimisation d'une fonction de perte (différentiable). Le Gradient Boosting combine essentiellement des "weak learners" en un seul "strong learner" de manière itérative. Chaque nouvel weak learner est entrainé sur les résidus de l'arbre précédent. À mesure que chaque weak learners est ajouté, un nouveau modèle est ajusté à l'aide d'un hyperparamètre γ pour fournir une estimation plus précise de la variable cible. L'idée du Gradient Boosting est qu'il peut combiner un groupe de modèles de prédiction relativement faibles pour construire un modèle de prédiction plus fort. Notre implémentation suit l'algorithme suivant:

Algorithm 2 Gradient Boosting Pseudocode

procedure GRADIENTBOOSTING(X, y)
  Choice of loss function: $L(y_i, F(x_i)) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$
  Initialize $F_0(x) = \bar{y}$
  for $t = 1$ to T **do**
    (1) Compute the negative gradient $r_i = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$ where L is the loss function
    (2) Fit a weak learner to the residuals r_i and obtain $f_t(x)$
    (3) Update the model: $F_t(x) \leftarrow F_{t-1}(x) + \gamma f_t(x)$ where γ is the learning rate
  end for
  return $F_T(x)$
end procedure

3.4 SVM

Le Support Vector Machine (SVM) est un algorithme de classification supervisée qui peut également être utilisé pour résoudre des problèmes de régression. Il fonctionne en trouvant une séparation hyperplan optimale entre l'espace des features en minimisant la distance entre les données les plus proches de chaque classe (appelées "vecteurs de support"). Les SVM peuvent également être utilisés pour résoudre des problèmes de classification non linéaires en utilisant des transformations de caractéristiques pour les projeter dans un espace de plus haute dimension où une séparation linéaire peut être trouvée (Astuce du Noyau). Nous utiliserons ce modèle dans notre projet cependant, pour des raisons de temps, il n'a pas été possible d'implémenter ce modèle dans ce projet sans l'usage des librairies.

3.5 Optimisation des Hyperparamètres

Dans l'optique d'optimiser les hyper-paramètres du 1er niveau des algorithmes du stacked-ANN, cinq méthodes sont proposées. Nous avons fait le choix de n'en implémenter seulement 2: le Circular Block Bootstrap et le Stationary Bootstrap. Chacun de ces modèles peuvent uniquement appliqué seulement sur les series temporelle stationnaire.

- *Circular Block Bootstrap:* Cette méthode [4], à l'identique du bootstrap, permet de générer un nouveau data set. Cependant, afin de garder l'aspect de corrélation dans la serie temporelle, les tirages aléatoire de données se font par block de donné de taille n fixe jusqu'à obtention d'un dataset équivalent en terme de taille au dataset original. C'est cette méthode qu'on a sélectionner pour l'optimisation des hyper-paramètres. Dans le training du ANN, on remarque clairement un overfitting sur certaine période du Stacked-ANN lorsqu'on utilise nos modèle propriétaire. La raison est que l'optimisation a seulement été faites sur 1 seule période tandis que les hyper-paramètres sont utilisé sur toutes les périodes. On note aussi l'impossibilité d'effectuer un Grid Search large, a cause de l'efficience des modèles.

- *Stationary Bootstrap:* Cette méthode [3] est similaire au Circular Clock Bootstrap mais introduit une notion d'aléatoire dans la taille des blocs de données tirés. La taille des blocs est donc aléatoire bien qu'elle suit une moyenne prédéfini. L'algorithme de construction des blocs a été implémenté par [1] en MATLAB, que nous avons traduit en Python.

Algorithm 3 Circular Block Bootstrap for the optimization of the hyperparameters

procedure CIRCULAR BLOCK BOOTSTRAP($X, y, hyperparameters, model, size, n^{iter}$)
Choice of loss function: $L(y_i, F(x_i)) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$
Create a list hp of the combination of all the hyperparameters
for $t = 1$ to n^{iter} **do**
(a) Create m block of data. Each block has the same length size.
(b) Pick randomly with replacement m times to create the new data named: bootstrap data
(c) Separate data into a test and train set
for hp_i in $\{hp_1, ..., hp_n\}$ **do**
(1) fit model with hyperparameters hp_i on the new bootstrap data
(2) predict on the test set of the bootstrap data
(3) compute the errors e_i and save them in a dictionary named *best*
end for
end for
$hp^* = \arg\min_{best} \frac{1}{n^{iter}} \sum_{i=1}^{n^{iter}} e_{k_i}$
return hp^*
end procedure

Algorithm 4 Stationary Bootstrap for the optimization of the hyperparameters

procedure STATIONARY BOOTSTRAP FOR THE OPTIMIZATION OF THE HYPERPARAMETERS($X, y, hyperparameters, model, size, n^{iter}$)
Choice of loss function: $L(y_i, F(x_i)) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$
Create a list hp of the combination of all the hyperparameters
for $t = 1$ to n^{iter} **do**
(a) Create bootstrap data (using algorithm 5)
(b) Separate data into a test and train set
for hp_i in $\{hp_1, ..., hp_n\}$ **do**
(1) fit model with hyperparameters hp_i on the new bootstrap data
(2) predict on the test set of the bootstrap data
(3) compute the errors e_i and save them in a dictionary named *best*
end for
end for
$hp^* = \arg\min_{best} \frac{1}{n^{iter}} \sum_{i=1}^{n^{iter}} e_{k_i}$
return hp^*
end procedure

Algorithm 5 Stationary Bootstrap

procedure STATIONARY BOOTSTRAP($X, m, length$)
  $accept = 1/m$
  lenData = number of rows in X
  $sampleLength = length$
  $sampleIndex$ = Uniform choice in the range (0,lenData-1)
  sample = an array of $lenData$ rows and the number of columns of X
  for $iSample = 0$ to $sampleLength$ **do**
    (a) Simulate a random value r using the uniform distribution
    if $r >= accept$ **then**
      (a.1) $sampleIndex \leftarrow sampleIndex + 1$
      if $sampleIndex >= lenData$ **then**
        (a.1.1) $sampleIndex = 0$
      end if
    else
      (a.2) $sampleIndex$ = Random number in the range 0 to lenData
    end if
    (b) $sample(iSample) \leftarrow X(sampleIndex)$
  end for
  return $sample$
end procedure

3.6 ANN

Le réseau de neurone mis en place dans notre implémentation suit les caractéristiques du papier. C'est à dire que 2 hidden layers sont construit avec respectivement 20 et 10 neurones. Afin d'estimer les paramètres, on les optimise en utilisant la technique d'optimisation ADAM (avec les paramètres $\beta_1 = 0.9$ et $\beta_2 = 0.999$) de manière itérative (avec 1000 epochs). Concernant la backward pass, cette dernière est faite sur la fonction de coût RMSE:

$$L(y_i, F(x_i)) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 + \lambda||w||^2} \tag{5}$$

D'autre part, une méthode de Grid Search est appliquée pour trouvé le paramètre de L2 régularisation et du learning rate.

3.7 Construction et Implementation

La première étape consiste à sélectionner et répartir soigneusement les datas. La répartition est la suivante :

- 25% des données sont utilisées pour entraîner le premier niveau d'algorithme (Random Forest, Gradient Boosting et SVM).
- 50% des données sont utilisées pour les estimations de ANN.
- 25% destinées pour la partie validation.

Les variables d'entrées de notre ANN sont :

- Les volatilités sont calculées en prenant la standard déviation des 5 derniers returns. 30 dernières volatilités normalisées entre 0 et 1.
- Les volatilités prédites avec les algorithmes du Random Forest, Gradient Boosting et SVM.

La dernière étape du modèle Stacked-ANN est l'ajustement de l'ANN, qui est l'algorithme empilant les prévisions faites par le RF, le GB et le SVM

Pour réaliser cela on a procédé en 4 temps :

- Construction des modèles Random Forest et Gradient Boosting puis comparaison avec la librairie sklearn. Le SVM n'a pas été implémenté.
- Le réseau de neurone a été implémenté en utilisant le framework de PyTorch.
- A partir de ces modèles, des prédictions ont été faites et les différents modèle ont été comparé entre eux.

4 Comparaison des résultats

La comparaison des modèles de référence avec le modèle proposé, le Stacked-ANN en termes de précision et de mesure du risque est effectuée avec un ensemble différent de données contenant les informations de l'année suivante (par exemple, si les données de 2000 à 2007 sont utilisées pour former et tester le modèle Stacked-ANN, les données hors échantillon sélectionnées à des fins de comparaison seront les mouvements du marché survenus en 2008). D'après le papier on s'attend à ce que le modèle Stacked-ANN soit le plus performant parmi les autres modèles, comme le montre le tableau ci-dessous extrait du papier de recherche.

Nous avons été confronté à un problème majeur. L'auteur du papier utilise le modèle de Heston en tant que référence, mais n'explique pas la méthodologie d'implémentation de ce modèle. Cela a rendu difficile la reproduction des résultats. Pour résoudre ce problème, nous été obligé de développer notre propre méthodologie.

Comme le papier, le modèle Stacked-ANN est le plus performant. Donc nos résultats sont cohérents avec ceux du papier.

4.1 LSTM

Afin de tenter de capturer mieux les dépendances de long terme du modèle, nous introduisons le LSTM. Ce dernier trouve son avantage dans sa capacité à capturer les dépendances de long terme grâce à sa mémoire interne. Cette mémoire interne est mise à jour à chaque étape du traitement d'une séquence en utilisant une portes d'oubli et d'entrée qui contrôlent ce qui est conservé et oublié. Les poids du modèle sont ajustés pour maximiser la performance pour la tâche en cours en utilisant l'information stockée dans la mémoire interne ainsi que les entrées actuelles. Nous avons donc implémenter un LSTM afin d'évaluer et de comparer les prédictions avec le Stacked-ANN.

4.2 Mesure de risques

Une mesure de risque est une fonction définie sur l'espace des variables aléatoires, et prenant ses valeurs dans R. Une mesure de risque est une application vérifiant les propriétés suivantes:

- invariante en loi : si $X \overset{L}{=} Y$, alors $R(X) = R(Y)$,
- croissante : si $X \geq Y$, alors $R(X) \geq R(Y)$,
- invariante par translation : si $k \in R$, alors $R(X + k) = R(X) + k$,
- homogène : si $\lambda \in R^+$, alors $R(\lambda X) = \lambda R(X)$,
- sous-additive : pour tous risques X et Y , $R(X + Y) \geq R(X) + R(Y)$,
- convexe : si $\beta \in [0, 1]$, alors $R(\beta X + (1 - \beta)Y) \leq \beta R(X) + (1 - \beta)R(Y)$

Par conséquent, pour chaque modèle de volatilité on calcule les mesures de risque suivante :

- VaR : La VaR est une mesure de risque de perte des investissements. Il estime combien un ensemble d'investissements pourrait perdre, compte tenu des conditions normales du marché, au cours d'une période donnée, telle qu'une journée.

- CVaR : connue sous le nom d'Expected Shortfall, est une mesure d'évaluation du risque qui quantifie l'importance du risque de queue d'un portefeuille d'investissement. La CVaR est calculée en prenant une moyenne pondérée des pertes "extrêmes" dans la queue de la distribution des rendements possibles, au-delà du seuil de la valeur à risque (VaR).

Chaque mesure de risque va être calculée et validée au moyen des tests suivants :

- Kupiec (1995) vérifie si le nombre d'excès de VaR est aligné avec le niveau de confiance choisi.
- Christoffersen et al. (1997). L'objectif de ce test est de valider que les excès de VaR sont indépendants, identiquement distribués et en ligne avec le niveau de confiance sélectionné.
- AS1 qui évalue la pertinence de la CVaR, en partant du principe que la VaR a déjà été testée et considérée comme correcte d'un point de vue statistique.
- AS2 valide la CVaR sans faire d'hypothèse sur la pertinence de la VaR.

5 Conclusion

Ce projet nous a permis de nous familiariser ou bien d'en apprendre plus sur des modèles de Machine Learning. Nous avons rencontré quelques difficultés dans le codage de l'ANN et du SVM. Concernant le SVM nous avons pris la décision d'utiliser une librairie.

Nous pensons que ce modèle peut être utile sur le marché des matières premières, afin de prédire les volatilités. En effet depuis quelques temps, il s'agit d'un marché extrêmement volatile où les prix varient fortement d'un extrême à un autre. Or comme mentionné dans le papier, le Stacked-ANN est très utile lorsque le prix varie fortement car un ANN seul s'avère imprécis.

References

[1] Gregor Fabjan. The stationary bootstrap: *https://github.com/qnity/stationary_bootstrap_matlab*. 2023.

[2] T.Hastie R.Tibshirani G.James, D.Witten. *An Introduction to Statistical Learning*.

[3] Romano Politis. The stationary bootstrap.

[4] Romano Politis. A circular block-resampling procedure for stationary data. purdue university. department of statistics. 1991.