TRANSFORMERS PARA GERAÇÃO DE SUPERFÍCIES SUAVES DE VOLATILIDADE IMPLÍCITA

Rafael Zimmer
Institute of Mathematical Science and Computing
University of São Paulo - Brazil
rafael.zimmer@usp.br

Adalton Sena Filho
Institute of Mathematical Science and Computing
University of São Paulo - Brazil
adaltonsena@usp.br

Eduardo Brito
Institute of Mathematical Science and Computing
University of São Paulo - Brazil
eduardobritocusp.br

6 de junho de 2024

ABSTRACT

Neste trabalho implementamos uma rede neural com arquitetura baseada em *Transformers* para otimização de valores P, Q, R para diversos modelos *SABR* paramétricos (*parametric stochastic* α, β, ρ) para geração de superfícies de volatilidade implicita no contexto de opções financeiras. A motivação por trás do trabalho é permitir a interpolação e extrapolação suave da superfície, ou seja, tornando-a derivável em todos os pontos. O desenvolvimento da arquitetura tem como principal contribuição solucionar o problema na escolha dos candidatos $\alpha, \rho, volvol$ cujos parâmetros P, Q, R otimizados gerem a superfície com menor oportunidade de arbitragem. Por fim, utilizaremos os valores históricos observados de $\alpha, \rho, volvol$ e a capacidade inata das arquiteturas de transformers para processar observações de sequências com tamanhos não fixos, ou seja, para volatilidades de contratos com data de vencimento variadas e não fixas e gerar as superfícies, comparando os resultados com métodos de interpolação clássicos (linear e *splines*).

Keywords First keyword · Second keyword · More

1 Introdução

1.1 Uma breve introdução à volatilidade implícita e a reconstrução de superfícies

A superfície de volatilidade implícita desempenha um papel crítico no contexto de contratos de derivativos financeiros, especificamente contratos de opções [Ludkovski, 2023]. Uma opção é um tipo derivativo, ou seja, um ativo negociado em relação a outro ativo, em que o comprador do contrato adquire a opção de realizar ou não a compra ou venda de um determinado ativo para com o vendedor do contrato na data futura. O cálculo de requisitos de margem até a precificação destes derivativos dependem de uma gama de valores, entre eles o tempo até o vencimento do contrato, o preço do ativo a ser negociado, e o preço do ativo a ser negociado no futuro [Black and Scholes, 1973]. Há também uma variável adicional para a precificação do contrato, chamada de volatilidade implicita (*IVOL*). Tal valor refere-se à volatilidade considerada, ou esperada, para o preço do ativo no futuro, e é essencial para o uso na equação de Black-Scholes Merton, para precificação de contratos de opções [Merton, 1973].

Ter acesso aos valores corretos de volatilidade implicita permite a investidores realizarem vendas pelos preços corretos e condizentes com os valores reais dos contratos no mercado, contudo, nem sempre é possível obter valores da *IVOL* que reflitam totalmente as características das ofertas existentes. Garantir uma forma de obter esse valor com confiança e com os dados disponíveis no mercado é uma tarefa essencial para analistas quantitativos [Kim et al., 2006]. Estratégias tradicionais para cálculo diário destes valores são fundamentadas predominantemente em algoritmos de interpolação [Homescu, 2011] ou modelos de equações diferenciais estocásticas [Jordan and Tier, 2011], ambos buscando aproximar para cada par de preço de vencimento (também chamada de maturidade) e preço do ativo no futuro (também chamada de *strike*) um valor correto para a volatilidade implicita. Essa abordagem permite visualizar os resultados do modelo no formato de uma superfície, onde os eixos **XY** são os pares de maturidade e *strikes* e o eixo **Z** é a *IVOL*, também chamada de superfície de volatilidade implicita (SVI).

Garantir previsões precisas da SVI é fundamental, pois até pequenos erros podem levar a precificações incorretas e consequentemente perda de lucro ou até retorno negativo (chamados nesse contexto de oportunidades de arbitragem) [Hagan et al., 2014]. Modelos tradicionais como interpolações lineares, polinomiais e *splines* geram oportunidades de arbitragem na SVI, e muitas vezes lutam para capturar efetivamente a dinâmica do mercado, além de carecer flexibilidade para extrapolação [Hagan et al., 2014].

1.2 Contribuições do Trabalho

Neste artigo, propomos uma abordagem para enfrentar os desafios de interpolação e extrapolação da SVI, reduzindo oportunidades de arbitragem. O método escolhido usa uma abordagem baseada em soluções de equações diferenciais estocásticas, em específico o modelo *Stochastic Alpha, Beta, Rho* (*SABR*). Serão utilizados os valores históricos das volatilidades implicitas observadas junto de um otimizador não-linear para obter os parâmetros que minimizem o erro da superfície.

A otimização dos parâmetros do modelo *SABR* é comumente realizada utilizando apenas um subconjunto dos pontos observados diariamente [Kim et al., 2021], devido ao fato de utilizar todos os pontos insere diversos *outliers* que não estão de acordo com os preços reais do derivativo, inserindo no modelo otimizado oportunidades de arbitragem. Para solucionar essa dificuldade, comumente os parâmetros são filtrados manualmente de acordo com critérios de analistas especialistas [Hagan et al., 2002], e um subconjunto dos pontos candidatos é utilizado para a otimização dos parâmetros. O uso de modelos capazes de receber como entrada nuvems de pontos (*point clouds* em inglês) permite filtrar pontos candidatos para serem utilizados pelos otimizadores não-lineares de forma muito mais rápida e automatizada. A contribuição principal desta pesquisa será no uso de arquiteturas de *transformers*, que ao contrário das redes neurais convencionais, aceitam quantidades variáveis de parâmetros para um determinado dia e diferentemente de redes neurais recorrentes é mais apropriado para lidar com nuvens de pontos não ordenadas por tempo [Kim et al., 2024]. Os processos de otimização e filtragem será abordado mais a fundo na seção de desenvolvimento.

1.3 A arquitetura do *Transformer*

O modelo *SABR* é um modelo de volatilidade estocástico utilizado frequentemente para obter superficies suaves, ou seja, contínuas e infinitamente deriváveis em todos os pontos [Hagan et al., 2015]. No contexto do modelo SABR, à cada um dos parâmetros candidatos será atribuido uma pontuação chamada de valor *SS*, que corresponde à aptidão dos candidatos em questão de serem resumidos ou representados por um subconjunto de pontos, ou seja, quão possível é reduzir a quantidade de candidatos sem alterar o formato da superfície. Em suma, esse valor de pontuação é normalizado, e utilizado para escolher apenas os pontos com maior pontuação, retirando no processo *outliers* que possam introduzir arbitragem no processo de otimização dos modelos *SABR*.

Usualmente, a escolha dos subconjuntos e o processo de pontuação é feito manualmente por analistas, como mencionado anteriormente, e a próxima seção focará nos detalhes abrangendo o modelo *SABR* e como utilizamos uma rede neural com atenção, ou seja, uma arquitetura de *Transformer*, especificamente a camada de ***encoder*** para a atribuição das pontuações de forma automática.

2 Fundamentação Teórica e Desenvolvimento

2.1 Formulação do Modelo SABR

O modelo SABR é definido por um conjunto de equações diferenciais estocásticas que descrevem a dinâmica do preço a termo e sua volatilidade. Os parâmetros do modelo incluem:

- **Alpha** (α): O nível inicial da volatilidade.
- **Beta** (β): O parâmetro de elasticidade que determina a dependência da volatilidade em relação ao preço do ativo.
- **Rho** (ρ): A correlação entre o preço do ativo e sua volatilidade.
- **Volvol** (ν): A volatilidade do processo de volatilidade.

A função de volatilidade implícita do modelo SABR é dada por:

$$\sigma_{BS}(f,K) = \alpha \frac{(fK)^{\frac{1-\beta}{2}}}{1 + \left(\frac{(1-\beta)^2\rho^2}{24} + \frac{(1-\beta)^2(2-3\rho^2)}{1920}\right)(fK)^{1-\beta}} \tag{1}$$

No contexto deste trabalho, serão gerados diversos modelos SABR diariamente, correspondendo à cada nível de maturidade diferente. Para tal, foi utilizado uma função contínua para mapear as diferentes maturidades para os parâmetros do modelo. São utilizadas 3 variáveis que codificam os níveis de maturidade, sendo elas:

- **P**: Dada uma função $f_\alpha(\cdot, \mathbf{P}) \to \mathbb{R}$, os valores α dos diversos modelos *SABR* são codificados dado um único vetor P, tal que $\mathbf{P} \in R^5$.
- **Q**: Dada uma função $f_\rho(\cdot, \mathbf{Q}) \to \mathbb{R}$, os valores ρ dos diversos modelos *SABR* são codificados dado um único vetor Q, tal que $\mathbf{Q} \in R^4$.
- **R**: Dada uma função $f_\nu(\cdot, \mathbf{R}) \to \mathbb{R}$, os valores ν dos diversos modelos *SABR* são codificados dado um único vetor R, tal que $\mathbf{R} \in R^4$.

As funções f_α, f_ρ, e f_ν são dadas de acordo com as expressões:

$$f_\alpha(t,p) = p_0 + \frac{p_3}{p_4}\left(\frac{1-e^{-p_4 t}}{p_4 t}\right) + \frac{p_1}{p_2}e^{-p_2 t} \tag{2}$$

$$f_\rho(t,q) = q_0 + q_1 t + q_2 e^{-q_3 t} \tag{3}$$

$$f_\nu(t,r) = r_0 + r_1 t^{r_2} e^{r_3 t} \tag{4}$$

2.2 Arquitetura Transformer Utilizada e Etapas de Ajuste de Parâmetros

Transformers são um tipo de arquitetura de rede neural conhecida por sua capacidade de capturar dependências complexas em dados sequenciais. Neste projeto, um transformer encoder é usado para ajustar os parâmetros do modelo SABR (P, Q e R), correspondendo a α, ρ e ν respectivamente. O transformer processa dados históricos de opções e produz estimativas de parâmetros que minimizam o erro entre as volatilidades implícitas do modelo e as volatilidades de mercado observadas.

Diferente das redes neurais recorrentes (RNNs) e das redes neurais convolucionais (CNNs), os transformers não dependem de uma estrutura sequencial para processar dados. Em vez disso, utilizam um mecanismo de atenção, que permite que cada elemento de entrada (ou token) considere a relação com todos os outros elementos na mesma sequência simultaneamente. Isso é realizado através de uma série de camadas chamadas de encoders (para codificação dos dados de entrada) e decoders (para gerar a saída).

Transformers são classificados como algoritmos de aprendizado de máquina devido à sua capacidade de aprender padrões e relações a partir de dados. Eles são treinados usando técnicas supervisionadas. No contexto da arquitetura específica que implementamos, utilizamos apenas as camadas do *encoder*, com 4 dimensões de entrada, 2 cabeças de atenção por camada e 4 camadas no total. A etapa de treino do modelo é realizada em cima de dados históricos de opções em cima do papel **S&P 500** para o ano de 2021-2022. As épocas de treino do modelo ajustam os pesos da camada com uma função de erro que minimiza a diferença entre suas previsões e os valores das pontuações geradas

manualmente. Essa capacidade de aprendizado é o que permite que transformers generalizem bem para novos dados não vistos durante o treinamento, fazendo previsões precisas baseadas nos padrões aprendidos.

A tarefa de ajuste de parâmetros do modelo SABR é especificamente uma tarefa de regressão. Em aprendizado de máquina, a regressão refere-se a problemas onde a saída é uma variável contínua, ao contrário da classificação, onde a saída é uma categoria discreta. No nosso caso, a saída do modelo transformer são as pontuações contínuas para cada parâmetro α, ρ, e ν usados como entrada do algoritm quasi-Newton de otimização não-linear Broyden-Fletcher-Goldfarb-Shanno (BFGS) [Broyden, 1970, Fletcher, 1970, Goldfarb, 1970, Shanno, 1970], tendo como alvo os valores **P, Q** e **R** que minimizem possíveis oportunidades de arbitragem de contratos com maturidades e *strikes* interpolados ou extrapolados.

2.3 Detalhes da Implementação

2.3.1 Classe do Modelo SABR

A classe 'SABRModel' implementa o modelo SABR, fornecendo métodos para calcular preços a termo, volatilidades implícitas e ajustar os parâmetros do modelo. Os métodos principais incluem:

- 'star()': Otimiza os valores P, Q e R que minimizem o erro da superfície gerada com a observada dado valores fixos para tempos de vencimento, preço à vista e futuro, taxa livre de risco e rendimento de dividendos e α, ρ, e ν variáveis.
- 'ivol()': Calcula a volatilidade implícita para parâmetros de modelo e condições de mercado dadas.

2.4 Arquitetura do Transformer

A arquitetura do transformer é conhecida por sua capacidade de capturar dependências complexas em dados sequenciais, utilizando um mecanismo de atenção para ponderar a importância de diferentes partes da sequência. A seguir, descrevemos a implementação de um modelo transformer em PyTorch, destacando os componentes principais: a atenção própria (Self-Attention), o bloco do transformer (Transformer Block) e o encoder do transformer (Transformer Encoder).

2.4.1 Atenção Própria

A classe `SelfAttention` implementa o mecanismo de atenção própria, que aceita dados contínuos e calcula a atenção para diferentes cabeças. A atenção é calculada através das matrizes de valor (`value`), chave (`key`) e consulta (`query`), resultando em uma saída ponderada pela importância de cada parte da sequência.

$$\textit{Attention}(Q, K, V) = \textit{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{5}$$

A implementação inclui:

```
class SelfAttention(nn.Module):
 def __init__(self, in_features, heads):

  self.head_dim = in_features // heads
  self.in_features = in_features
  self.heads = heads

  self.W_v = nn.Linear(self.head_dim, self.head_dim, bias=False)
  self.W_k = nn.Linear(self.head_dim, self.head_dim, bias=False)
  self.W_q = nn.Linear(self.head_dim, self.head_dim, bias=False)

  self.fc_out = nn.Linear(in_features, in_features)

 def forward(self, value, key, query, mask=None):
  N = query.shape[0]
  value_len, key_len, query_len = value.shape[1], key.shape[1], query.shape[1]

  value = value.reshape(N, value_len, self.heads, self.head_dim)
```

```
  key = key.reshape(N, key_len, self.heads, self.head_dim)
  query = query.reshape(N, query_len, self.heads, self.head_dim)

  values = self.W_v(value)
  keys = self.W_k(key)
  queries = self.W_q(query)

  energy = torch.einsum("nqhd,nkhd->nhqk", [queries, keys])
  if mask is not None:
  energy = energy.masked_fill(mask == 0, float("-1e20"))

  attention = torch.softmax(energy / (self.head_dim ** (1 / 2)), dim=3)
  out = torch.einsum("nhql,nlhd->nqhd", [attention, values]).reshape(N, query_len, self.in_features)

  out = self.fc_out(out)
  return out
```

2.4.2 Bloco do Transformer

A classe `TransformerBlock` implementa um bloco do transformer, composto por um mecanismo de atenção própria, normalização em camadas (LayerNorm) e uma rede feed-forward. Este bloco é a unidade básica que será repetida várias vezes no encoder:

```
class TransformerBlock(nn.Module):
 def __init__(self, in_features, heads, forward_expansion):
  self.attention = SelfAttention(in_features, heads)
  self.norm1 = nn.LayerNorm(in_features)
  self.norm2 = nn.LayerNorm(in_features)

  self.feed_forward = nn.Sequential(
   nn.Linear(in_features, forward_expansion * in_features),
   nn.ReLU(),
   nn.Linear(forward_expansion * in_features, in_features)
  )

 def forward(self, x, mask=None):
  attention = self.attention(x, x, x, mask)
  x = self.norm1(attention + x)
  forward = self.feed_forward(x)
  out = self.norm2(forward + x)
  return out
```

2.4.3 Encoder do Transformer

A classe `TransformerEncoder` implementa o encoder do transformer, composto por vários blocos do transformer. Este encoder processa a sequência de entrada, aplicando múltiplas camadas de atenção própria e redes feed-forward:

```
class TransformerEncoder(nn.Module):
 def __init__(self, in_features, heads, num_layers, forward_expansion, dropout, out_features):
  self.layers = nn.ModuleList([
   TransformerBlock(in_features, heads, forward_expansion) for _ in range(num_layers)
  ])

  self.dropout = nn.Dropout(dropout)
  self.fc_out = nn.Linear(in_features, out_features)

 def forward(self, x, mask=None):
  for layer in self.layers:
   x = layer(x, mask)
  return self.fc_out(x)
```

3 Metodologia e Resultados

3.1 Coleta de Dados

Os dados utilizados neste trabalho foram obtidos de dados históricos de opções sobre o ativo SPY, da bolsa estadunidense, de 2021 à 2022, disponível no Kaggle. Este conjunto de dados contém diversas colunas sobre opções negociadas, indexadas pela data, incluindo preços de exercício, preços futuros, maturidades, volatilidades implícitas, entre outros []. O arquivo pode ser e foi baixado e armazenado localmente em um diretório específico para processamento.

3.2 Descrição do Conjunto de Dados

O conjunto de dados contém as seguintes colunas relevantes para nosso estudo, além de diversas outras:

- **underlying**: Preço do ativo subjacente.
- **strike**: Preço de exercício da opção.
- **daysToExpiration**: Dias restantes até a expiração da opção.
- **iv**: Volatilidade implícita.
- **dt**: Data do registro.

Além disso, adicionamos duas colunas calculadas:

- **maturity**: Calculada dividindo *daysToExpiration* por 360 (dias úteis no ano) para normalizar o tempo de maturidade.
- **r**: Taxa livre de risco obtida do Federal Reserve Economic Data (FRED) para o período correspondente.
- **d**: Taxa de dividendos paga trimestralmente pelo SPY.

3.3 Pré-processamento dos Dados

O pré-processamento dos dados encorreu em várias etapas essenciais, predominantemente realizadas para permitir o uso dos dados diretamente na arquitetura do transformer e subsequente uso para gerar a superfície para os dados históricos. O transformer tem como entrada 4 dimensões, sendo elas:

1. O tempo até a maturidade;
2. O valor do parâmetro (α, ρ, ou ν);
3. **1** se o parâmetro foi calculado para o dia atual, **0** se foi calculado no dia anterior por não existir no dia atual;
4. Parâmetros calculados utilizando as pontuações do dia anterior.

A variável target são os scores gerados utilizando-se todos os pontos candidatos para o dia atual, e a função de erro do modelo utiliza uma porcentagem **p** de pontos e é treinado para minimizar a função de erro quadrática das pontuações.

3.4 Métricas de Erro e Comparação das Superfícies

Utilizamos como baseline uma interpolação linear dos pontos observados diariamente, e comparamos com a performance do modelo em relação à pontos internos à uma malha (ou *grid* em inglês) gerada para pontos dentro do intervalo diário, tanto para valores de maturidade como de *strike*.

O nosso modelo transformer foi treinado em cima do conjunto de dados, de aproximadamente 70 observações, cada uma com uma sequência de em média 700 pontos. Os modelos foram treinados por aproximadamente 700 épocas.

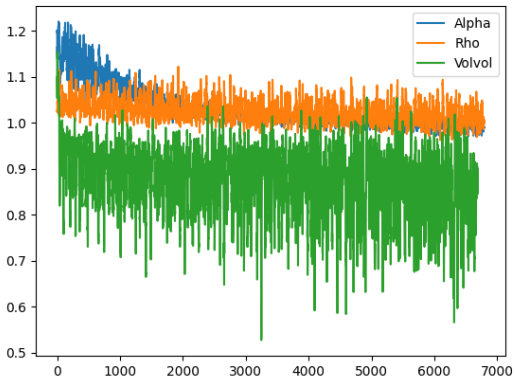

Figura 1: Função de perda histórica, de uma sessão de treinamento do Transformer

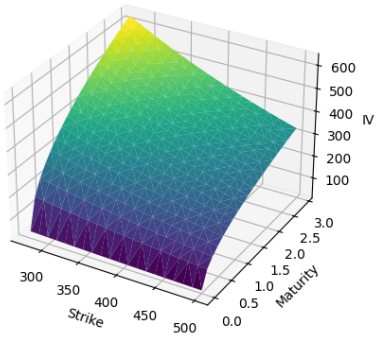

Figura 2: Superfície gerada utilizando as pontuações retiradas do Transformer e dos modelos SABR otimizados com os novos candidatos

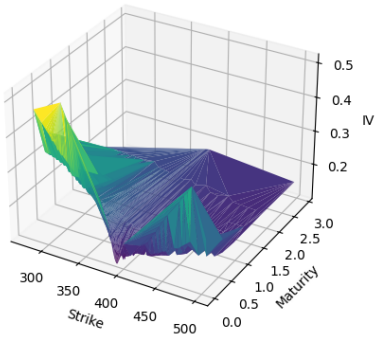

Figura 3: Superfície interpolada linearmente, percebe-se grande rigidez nos pontos intermediários

4 Conclusão

4.1 Resumo

Este trabalho apresentou uma abordagem para ajustar os parâmetros do modelo SABR utilizando uma rede neural baseada na arquitetura de transformers (especificamente o elemento do *encoder*). A combinação do modelo SABR com a arquitetura transformer foi utilizada como forma de replicar os padrões da superfície de volatilidade implicita dia à dia, gerando a superfície suave e sem oportunidade de arbitragem para os contratos existentes observados no dia. O fato da escolha e pontuação dos candidatos ser possível de ser realizada com um modelo automático, em vez de manualmente, é uma solução extremamente eficiente, principalmente se considerarmos a possibilidade de utilizar esses modelos na GPU, o que permite a reconstrução da superfície em instantes, permitindo investidores identificar oportunidades de negociação em apenas poucos instantes, em vez de em uma escala diária.

4.2 Implicações e Trabalhos Futuros

O principal foco do trabalho é realmente a abordagem utilizando o transformer para substituir a escolha manual das pontuações para os candidatos, abrindo caminho para eventuais novas pesquisas e estratégias de High-Frequency Trading (ou *HFT*, abreviação em inglês). Pesquisas futuras podem explorar a integração de fatores de mercado adicionais e aprimorar a arquitetura que escolhemos, possivelmente aumentando a quantidade de blocos ou cabeças de atenção. Além disso, é importante considerar a escolha do otimizador dos parâmetros **P, Q, R**, pois o tempo para realizar a etapa de preprocessamento foi um grande limitante no decorrer do trabalho. Estender o modelo para outros instrumentos financeiros também é uma possibilidade, pois permite o estudo sobre a adaptação do modelo a diferentes regimes de mercado e condições econômicas (tanto micro- como macro-econômicas) variáveis podem revelar novas oportunidades de aplicação e desenvolvimento do método proposto.

Referências

M. Ludkovski. Statistical machine learning for quantitative finance. *ANNUAL REVIEW OF STATISTICS AND ITS APPLICATION*, 10:271–295, 2023. ISSN 2326-8298. doi:10.1146/annurev-statistics-032921-042409.

Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81 (3):637–654, 1973.

Robert C Merton. Theory of rational option pricing. *Bell Journal of Economics and Management Science*, 4(1): 141–183, 1973.

Bo-Hyun Kim, Daewon Lee, and Jaewook Lee. Local volatility function approximation using reconstructed radial basis function networks. In J Wang, Z Yi, JM Zurada, BL Lu, and H Yin, editors, *ADVANCES IN NEURAL NETWORKS - ISNN 2006, PT 3, PROCEEDINGS*, volume 3973 of *Lecture Notes in Computer Science*, pages 524–530, HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY, 2006. Univ Elect Sci & Technol China; Chinese Univ Hong Kong; Asia Pacific Neural Network Assembly; European Neural Network Soc; IEEE Circuits & Syst Soc; IEEE Computat Intelligence Soc; Int Neural Network Soc; Natl Nat Sci Fdn China; KC Wong Educ Fdn, Hong Kong, SPRINGER-VERLAG BERLIN. ISBN 3-540-34482-9. 3rd International Symposium on Neural Networks (ISNN 2006), Chengdu, PEOPLES R CHINA, MAY 28-31, 2006.

Cristian Homescu. Implied volatility surface: Construction methodologies and characteristics, 2011.

Richard Jordan and Charles Tier. Asymptotic approximations to cev and sabr models. *SSRN Electronic Journal*, pages 1–38, May 2011. Available at SSRN: https://ssrn.com/abstract=1850709 or http://dx.doi.org/10.2139/ssrn.1850709.

Patrick S. Hagan, Deep Kumar, Andrew Lesniewski, and Diana Woodward. Arbitrage-free sabr. *Wilmott*, 2014(69): 60–75, 2014. doi:https://doi.org/10.1002/wilm.10290. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/wilm.10290`.

Hyeonuk Kim, Kyunghyun Park, Junkee Jeon, Changhoon Song, Jungwoo Bae, Yongsik Kim, and Myungjoo Kang. Candidate point selection using a self-attention mechanism for generating a smooth volatility surface under the sabr model. *EXPERT SYSTEMS WITH APPLICATIONS*, 173, JUL 1 2021. ISSN 0957-4174. doi:10.1016/j.eswa.2021.114640.

Patrick Hagan, Deep Kumar, Andrew Lesniewski, and Diana Woodward. Managing smile risk. *Wilmott Magazine*, 1: 84–108, 01 2002.

Soohan Kim, Seok-Bae Yun, Hyeong-Ohk Bae, Muhyun Lee, and Youngjoon Hong. Physics-informed convolutional transformer for predicting volatility surface. *QUANTITATIVE FINANCE*, 24(2):203–220, JAN 9 2024. ISSN 1469-7688. doi:10.1080/14697688.2023.2294799.

Patrick Hagan, Andrew Lesniewski, and Diana Woodward. *Probability Distribution in the SABR Model of Stochastic Volatility*, volume 110, pages 1–35. 06 2015. ISBN 978-3-319-11604-4. doi:10.1007/978-3-319-11605-1_1.

Charles G Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

Roger Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13(3):317–322, 1970.

Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24 (111):647–656, 1970.