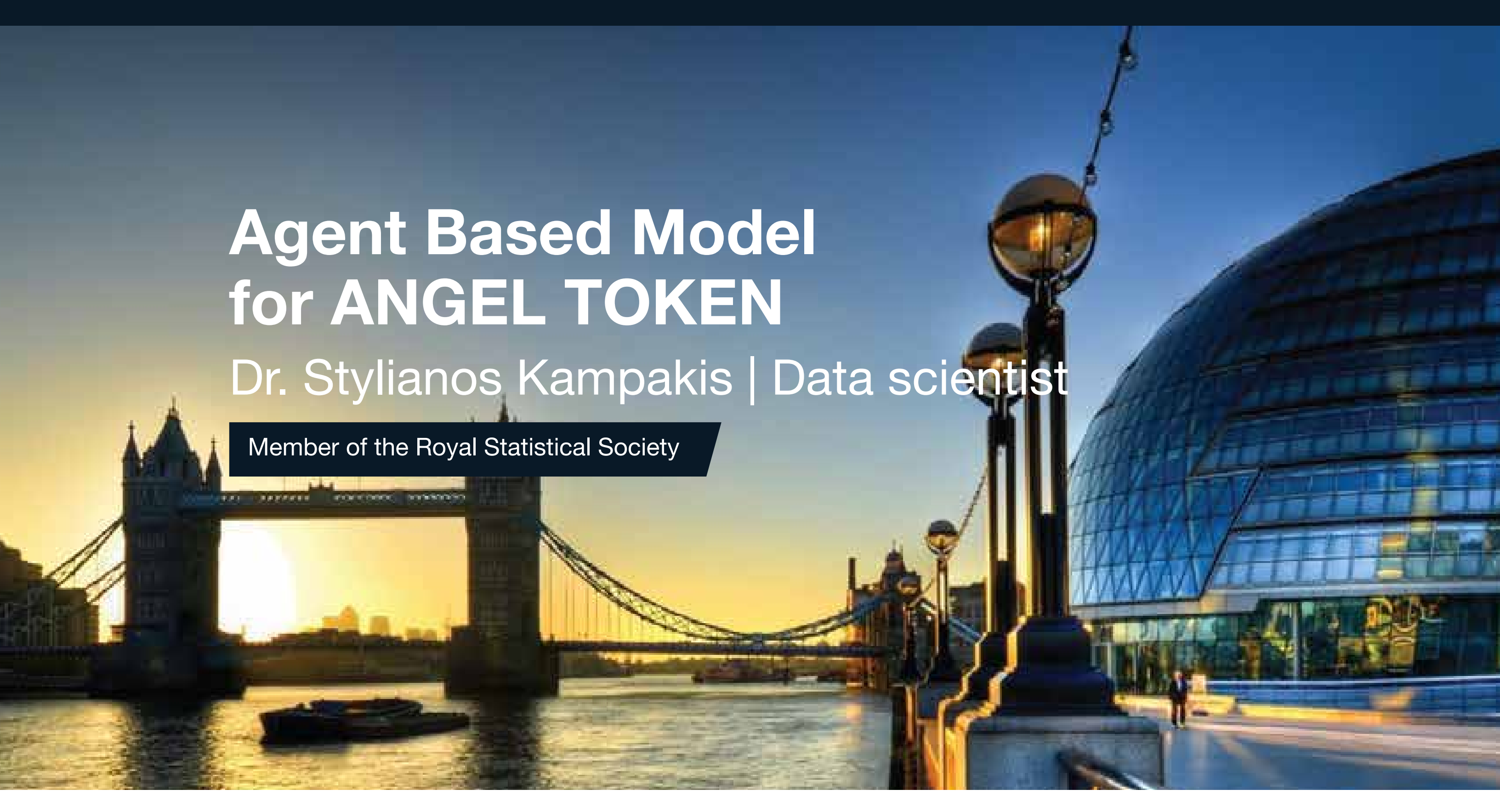


Agent Based Model for ANGEL TOKEN

Dr. Stylianos Kampakis | Data scientist
Member of the Royal Statistical Society

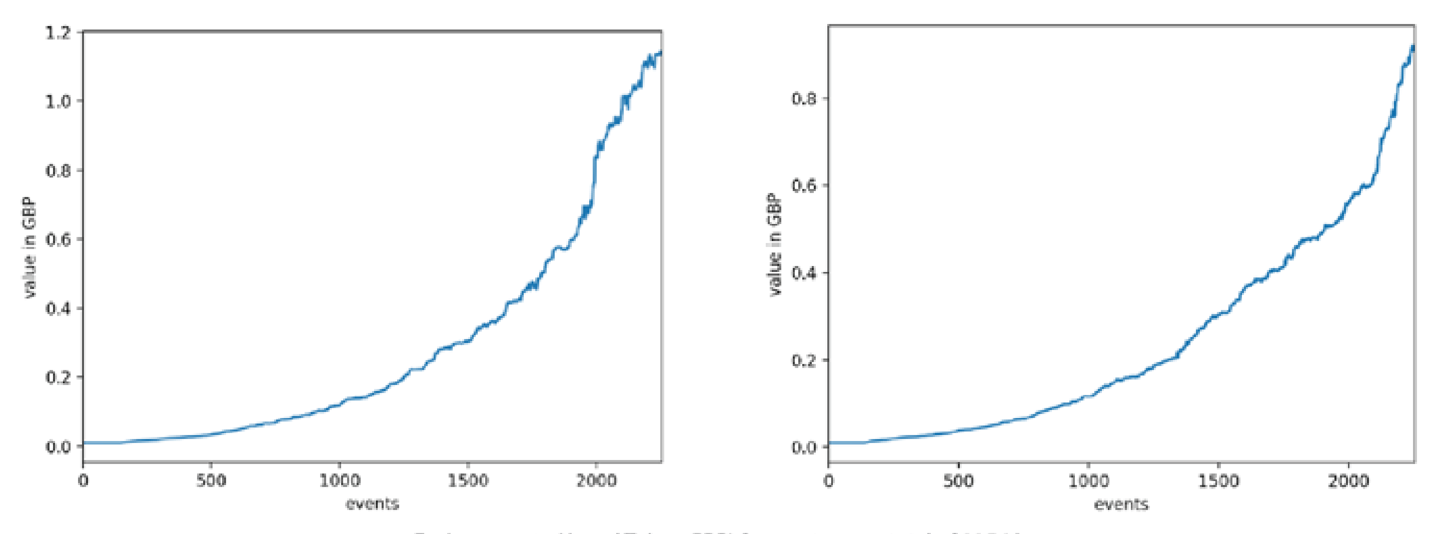
Agent based model overview

In order to study the tokenomics of Crowd for Angels an agent based modelling was employed. The agent based model allows an exploration of the interaction of the different parameters down to individual events, and the explicit modelling of variance induced by exogenous factors. Therefore, it is a robust way of forecasting the future value of the Angel Token.

The assumptions behind the model are the following:

- Time is measured in discrete events. These events are the loans handed out by CfA.
- The number and size of loans has been based on forecasts and calculations by CfA. They come to a total of approximately 2250 loans, with a value that could go up to £1 billion.
- At any given event, only a percentage of the users are active.
- The investment by the users follows a power law distribution.
- If the supply by the active users is not enough to satisfy demand, then the price of the tokens increases.
- The exchange rate is also influenced by speculation. This is simulated by applying Gaussian noise to the exchange rate.

Based on this information, the conclusion is that the exchange rate tends to reach parity with the GBP.
The plots below have been based on a1000 simulations.



Exchange rate (Angel Token:GBP) forecast over a total of 2256 loans and 1000 simulations

Technical details overview

The following parameters were used for the model:

Number of users: 1500
Initial exchange rate (Angel Token to GBP): £0.01
Loan fee (to be paid in tokens): 1%
Number of active users (at any given event): 10%
Exchange rate noise: Normal with mean and variance equal to the exchange rate.
Distribution of investment: (Power Law with coefficient 0.1 times £95k)+£5k

Justification behind the parameters:

Number of users: Given by CfA forecasts
Initial exchange rate: A reasonable assumption for a newly traded coin is that it will be traded according to the lowest common denominator of the market.
Loan fee: Fixed by CfA
Number of active users: This is based on an assumption that most users will be hoarders, who will just wait for the coin to rise in value.
Exchange rate noise: This is based on proprietary research on bitcoin, that showed that the variance of the fluctuations are correlated to the actual price. Given that there is enough liquidity in the market, it is reasonable to expect that this could happen with Angel Token as well.
Distribution of investment: A power law was empirically fit, based on consultation with CfA from prior investments.

Description of the algorithm

Create a list of loans L
Create a list of token holders U
Do until all loans are exhausted:
  Choose 1 loan
  Request the 1% fee in tokens for the loan
  Add noise to the exchange rate according to 10% probability.
  Choose the active pool of users
  If the users have do not have enough tokens, then select a new pool of users and increase the exchange rate to satisfy demand
  Run the lottery method for user selection
    Choose a random user who sells their tokens
    If demand is not fully satisfied move to another random user

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

n_users = 1500
#Angel Token to pound exchange
token_to_pound = 0.01
#the rate that the company must pay back in fees (in GBP)
rate_pounds=0.01
active_users_percentage=0.1

def user_lottery(users_starting,amount):
  k=True
  users=users_starting.copy()

  while k:
    winner = np.random.randint(0,len(users))

    if users[winner]>=amount:
      users[winner]=users[winner]-amount
      k=False
    else:
      amount=amount-users[winner]
      users[winner]=0

  return users

def run_sim(users_tokens,companies_round,token_to_pound,rate_pounds,
      active_users_percentage=active_users_percentage,noise_chance=0.01):
  initial_users = users_tokens.copy()
  iteration=0
  rates=[]
  for comp in companies_round:

    if np.random.rand()<noise_chance:
      dummy =
token_to_pound+np.random.normal(token_to_pound,token_to_pound/1)*np.sign(np.random.nor
mal())
      print(dummy)
      if dummy>0.01:
        token_to_pound=dummy

    pounds_required = rate_pounds*comp
    tokens_required = pounds_required/token_to_pound

    indices = np.random.permutation(len(users_tokens))
    indices = indices[:int(len(users_tokens)*active_users_percentage)]
    active_users = users_tokens[indices]

    if tokens_required>active_users.sum():
      if active_users.sum()==0.0:
        active_users = users_tokens
        indices=np.arange(0,len(users_tokens))

      token_to_pound=pounds_required/active_users.sum()
      tokens_required = pounds_required/token_to_pound
    else:
      active_users = user_lottery(active_users, tokens_required)
      users_tokens[indices]=active_users

    iteration+=1
    rates.append(token_to_pound)

  profit=(initial_users - users_tokens)*token_to_pound

  return initial_users,users_tokens,profit, token_to_pound,rates


profits = []
users=[]
pti=[]
rates=[]
rates_h=[]
for i in range(0,1000):
  print('running iteration :'+str(i))
  k=np.random.power(0.1,n_users)*95000+5000
  users_tokens_initial = k*95
  companies_y1 = [250000]*100+[500000]*30+[1000000]*10
  companies_y2 = [250000]*120+[500000]*40+[1000000]*13+[2000000]*1
  companies_y3 = [250000.0]*150+[500000.0]*65+[1000000]*18+[2000000]*12
  companies_y4 = [250000]*200+[500000]*100+[1000000]*52+[2000000]*20
  companies_y5 = [250000]*600+[500000]*400+[1000000]*200+[2000000]*125

companies_round=np.array(companies_y1+companies_y2+companies_y3+companies_y4+com
panies_y5)


  initial_users,users_tokens,profit,final_token_to_pound,rates_history =
run_sim(users_tokens_initial.copy(),companies_round,

token_to_pound,rate_pounds,active_users_percentage=active_users_percentage)

  profits.append(profit)
  print("\ninitial token distribution amongst users")
  print(initial_users)

  print("\nfinal token distribution after {0} investment rounds".format(len(companies_round)))
  print(users_tokens)
  users.append(users_tokens_initial)

  print("\nTotal profit per user in GBP is "+str(profit))
  print("\nAverage profit is "+str(np.mean(profit)))

  profit_to_initial=profit/users_tokens_initial
  pti.append(profit_to_initial)

  print("\nFinal rate is "+str(final_token_to_pound))
  rates.append(final_token_to_pound)

  rates_h.append(rates_history)


k=pd.DataFrame(rates_h)
k.median().plot()
plt.xlabel('events')
plt.ylabel('value in GBP')
```

Risk Warning