

# Relatório T1 - Modelagem e Simulação

## Membros:

- Manoella Rockenbach, 19102193
- Rodrigo Ferraz Souza, 19103563

## 1 - Pesquisa

### ***Geração de números pseudo aleatórios por relação recursiva***

A geração de números pseudo aleatórios geralmente ocorre por uma relação recursiva entre os valores. Relação na qual o próximo número da sequência é uma função do último ou dos dois últimos números gerados. Ou seja, dado um valor inicial  $x$ , chamado de semente, escolhido para dar início a sequência de números, o próximo valor é gerado da aplicação da função criada no último número gerado.

Uma das características desse método é ser determinístico, ou seja, conhecendo a função  $f$ , podemos gerar a mesma sequência de números pseudo aleatórios sempre que fornecermos o mesmo valor de semente. Além disso, os números gerados são periódicos, e a sequência acabará se repetindo eventualmente.

Esse método é útil se você precisar repetir a mesma sequência de números novamente. E é eficiente para produzir muitos números em um curto espaço de tempo.

### ***Métodos de geração:***

#### **Método congruente:**

$$x_n = a^n \bmod m$$

Desenvolvido pelo Prof. D. H. Lehmer, em 1951. Baseia-se na ideia que os restos de sucessivas potências de um número possuíam boas características de aleatoriedade. Sua função obtinha o  $n$ -ésimo número de uma sequência, tomando o resto da divisão da  $n$ -ésima potência de um inteiro  $a$  por um outro inteiro  $m$ .

#### **Método Congruente Multiplicativo:**

$$x_n = ax_{n-1} \bmod m$$

O parâmetro  $a$  é chamado de multiplicador e o parâmetro  $m$  de módulo. Os valores escolhidos por Lehmer para estas variáveis foram  $a = 23$  e  $m = 108 + 1$ , valores escolhidos pela facilidade de implementação no ENIAC, que era uma máquina de oito dígitos decimais.

### **Método Congruente Linear:**

$$x_n = (ax_{n-1} + b) \bmod m$$

Os valores de  $x_n$  são inteiros entre 0 e  $m-1$ , as constantes  $a$  e  $b$  são positivas. De maneira geral, a escolha dos valores de  $a$ ,  $b$ , e  $m$  afetam o período e a auto correlação na sequência.

### **Método de quadrados médios:**

Esse método inventado por John Von Neumann, tem uma vantagem em relação aos métodos anteriores pois a quantidade de números gerados antes que a sequência comece a repetir não depende mais do módulo da função usada para gerar a sequência de números aleatórios, dessa forma a extensão de números até as repetições começarem é maior.

Neste método, a partir de uma semente, esse número é então elevado ao quadrado, e os dígitos do centro são usados como próximo elemento da sequência, repetindo o processo. Caso o número de dígitos que fique à esquerda seja maior que os que ficam à direita não há problema, simplesmente fixamos para qual lado vamos fazer o corte.

A aleatoriedade da sequência e o seu período dependem apenas da semente inicial.

## **2 - Função desenvolvida**

### ***Descrevendo o algoritmo***

A técnica que utilizamos para fazer um programa que gere números pseudo aleatórios foi:

- Usamos o timestamp do momento da execução como seed
  - O primeiro da seed valor é 1 nos testes

```
seed = seed + ((float(time.time())))/seed
```

- Geramos uma hash em SHA256 da conversão da seed em string
- Somamos cada dígito em hexadecimal para obtermos a soma de todos os números.
- Então inicializamos a variável result

```
result = 1.01 + seed
```

- Com a soma em mãos executamos a seguinte operação para cada letra da hash gerada

```
result = hexstr_to_dec(l)*sum + result/sum
```

- Então retorna-se

```
int(result*10000 * seed**2)
```

Testando para avaliar sua aleatoriedade

Para analisar bem este problema foi utilizado duas técnicas, que também serão utilizadas no algoritmo tradicional discutido mais abaixo para fins de comparação. E são elas:

- Gerar números infinitamente e guardá-los em uma lista e, caso seja gerado um repetido, o loop é interrompido e mostra-se em tela quantas iterações se passaram até que tal fato ocorresse.
  - Nos nossos testes o algoritmo gera, em média, mais de 10 000 000 números antes de repetir
- Gerar números aleatórios entre 0 e 255 e inseri-los em uma matriz 1920x1080 e exibir em tela a imagem em escala de cinza que isto representa, mostrando um ruído, para avaliarmos, graficamente, a existência, ou não, de algum padrão na geração de números
  - O resultado do nosso algoritmo é o que segue:

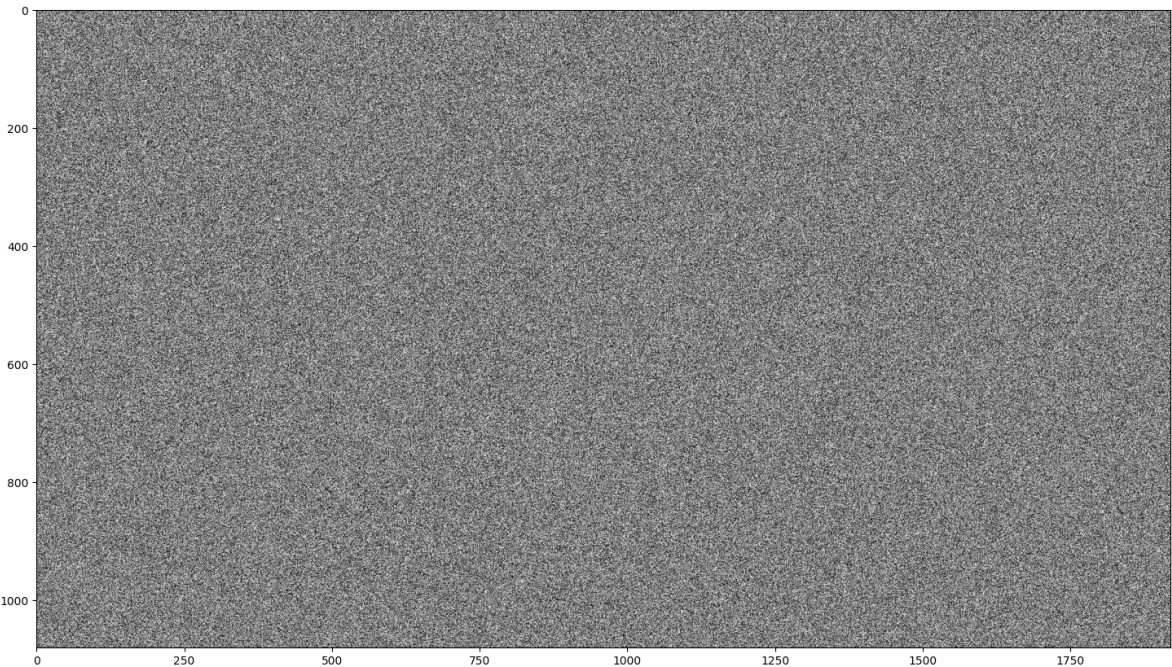


Tabela de números aleatórios

Table of random Numbers									
71625216	58920960	12708608	46790912	19842304	22092032	89723904	44460800	25578240	89303808
64494336	27912704	05607936	06818816	35761408	07901440	45508864	89186304	54319872	56304128
09362688	14270464	61636608	79773184	09547520	50213632	85314816	72028416	79479552	65274880
22857216	76155136	60060928	61983744	97699072	25860352	07942912	47660032	32594432	96043264
95585024	22152960	91574016	71457024	41858560	43540992	26468096	23044864	36399616	45299200
17320704	19283712	17276672	70327552	88512768	15933696	71881728	61341952	00284416	64909568
66936576	44321280	38432256	46233088	48948224	05181440	06632960	90914048	04675840	17055232
82858752	83826176	87471616	56776192	56467968	71696128	76441344	68881664	63358720	92221696
67897600	73821696	81658368	72228608	93589248	29381632	12300544	89442048	14242048	80822528
45810432	76055040	99499264	01275392	52206336	59083776	39735552	77112576	66392064	60425984

## 3 - Função dos quadrados médios

### *Descrevendo o algoritmo*

A parte inteira do timestamp, um número de 10 dígitos, é usado como a semente inicial do algoritmo. A partir disso, a semente é elevada ao quadrado e são escolhidos os dez números do meio como a próxima semente. Isso é feito sucessivamente sempre que a função é chamada.

```
import math
import time

seed = int(time.time())
seed_size = len(str(seed))
start = math.floor(seed_size/2)
end = start + seed_size
def mid_square():
    global seed
    global start
    global end

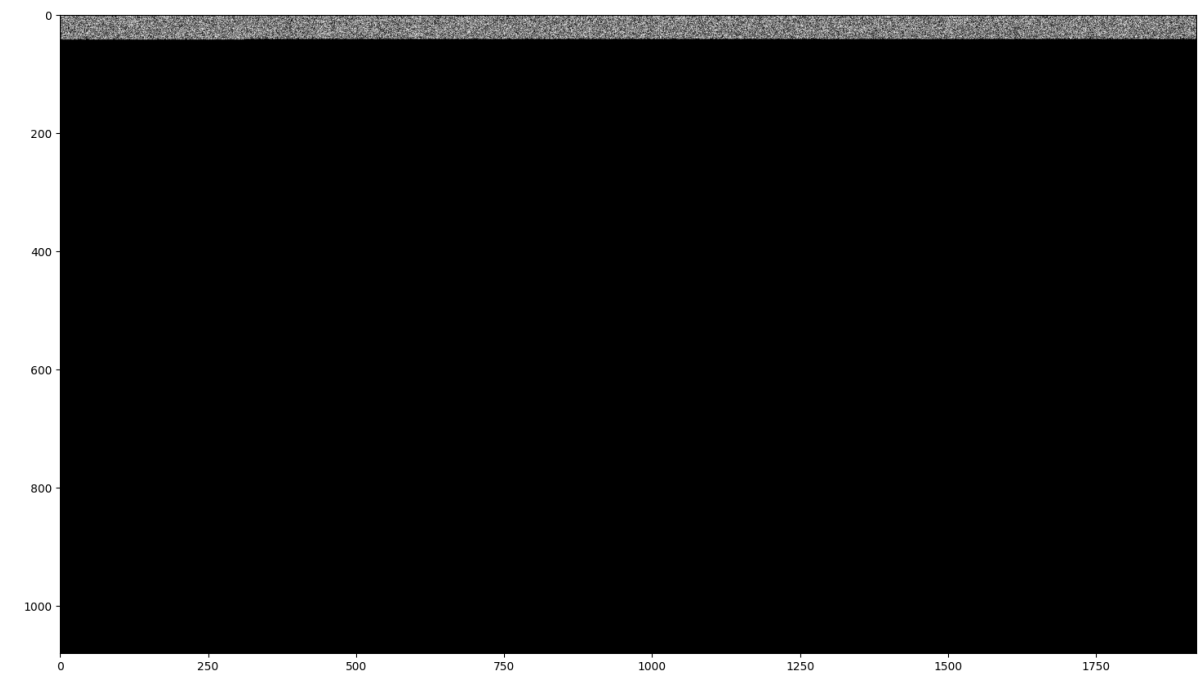
    seed_number = seed
    seed_number=int(str(seed_number*seed_number).zfill(seed_size*2)[start:end])
    seed = seed_number
    return seed_number
```

### *Testando para avaliar sua aleatoriedade*

Os mesmos testes que foram feitos no tópico anterior foram retomados aqui, para que possamos comparar a performance dos códigos.

Os testes foram feitos utilizando uma seed gerada com o timestamp do momento do teste, portanto tinham 10 dígitos.

- Gerando números até que um repetido seja gerado.
  - em média foram geradas 500 números até que um fosse repetido
- Gerando o noise de 1920x1080



***Tabela de números aleatórios***

Table of random Numbers									
88836947	41713522	14015176	23051583	99236788	79400925	00468908	09782747	94371388	59588730
79567430	97799168	28468614	85035830	85423837	95543278	03767709	54464311	80577727	80412884
17615131	43670401	31189235	60803798	78698512	70877910	93121259	48710777	87055959	95217974
05817726	27443358	35818983	01513431	67592733	09891544	93866427	28753177	42337875	58206595
06747014	37305979	24982691	71976496	84815764	73114229	50552822	80046121	13530871	44846700
66065008	17116820	69015269	38827551	08439166	23807227	58098574	20219008	67378845	79637535
01479808	19978317	09763501	66297517	59253603	40486684	47579813	50710051	99580724	40493923
51729999	79763965	34621125	04472962	79157890	22475492	20173406	29555096	05788995	76234631
02023637	93747067	86515711	57938498	99683504	05097697	49321147	98161413	14232021	74848217
22707880	38478140	37312578	30396770	55396264	81108651	22494670	80401784	83428703	20138842