Programmation GPU: Nested Monte Carlo vs. regression

Alexander Baumann & Patrick Lutz — 18.03.2022

Cadre

Problématique:

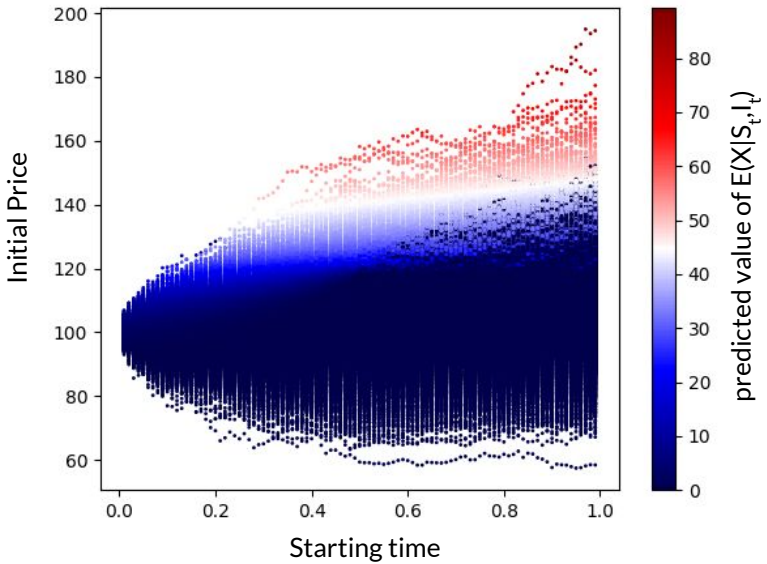
Étant donné un couple (S_t, I_t) à un instant t, estimer $E(X|S_t, I_t)$ où $X = (S_T - K)_+ 1_{\{I(t) \in [P1, P2]\}}$.

Méthodes proposées:

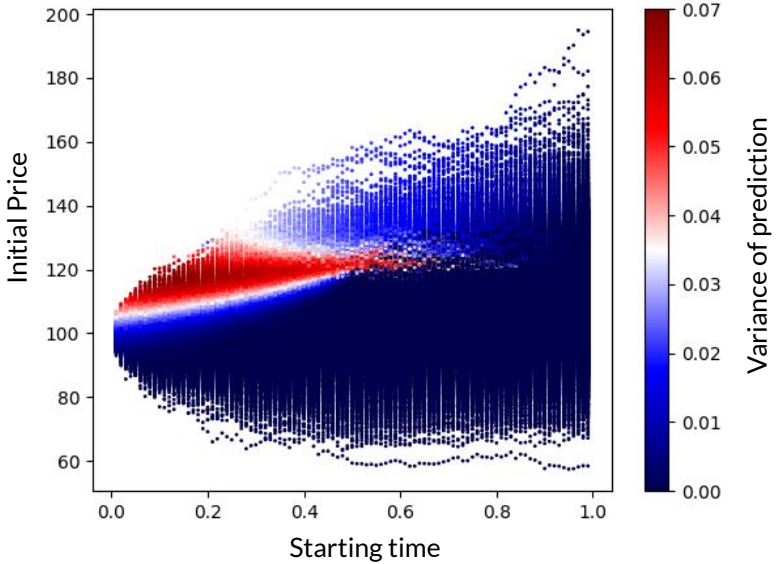
1. Simulation Monte Carlo imbriquée
2. Inférence d'un modèle linéaire entraîné au préalable
3. Inférence d'un réseau de neurones entraîné au préalable

Résultats: Monte Carlo imbriqué

Valeurs de Monte Carlo imbriqué

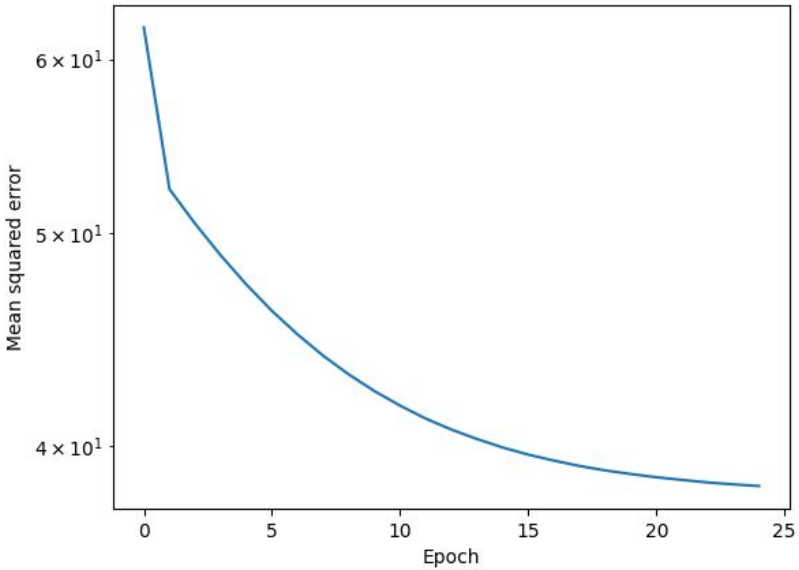


Variance de Monte Carlo imbriqué

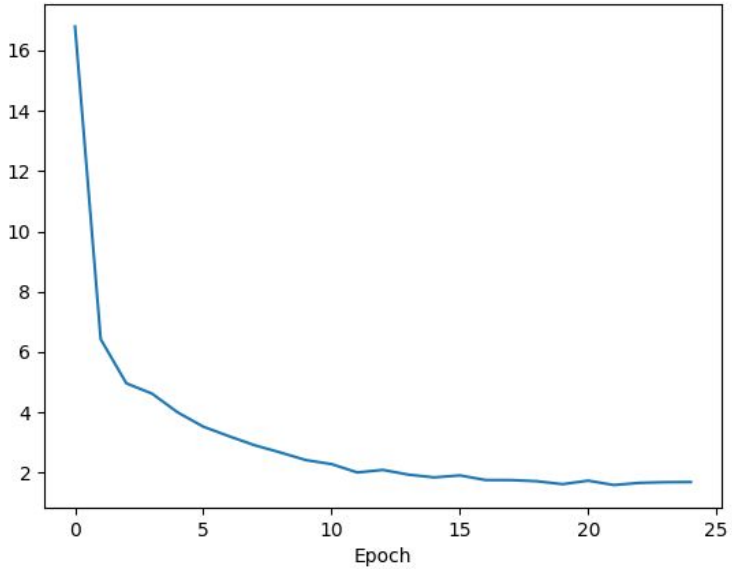


Convergence: Régression lin. et réseau de neurones

Régression linéaire

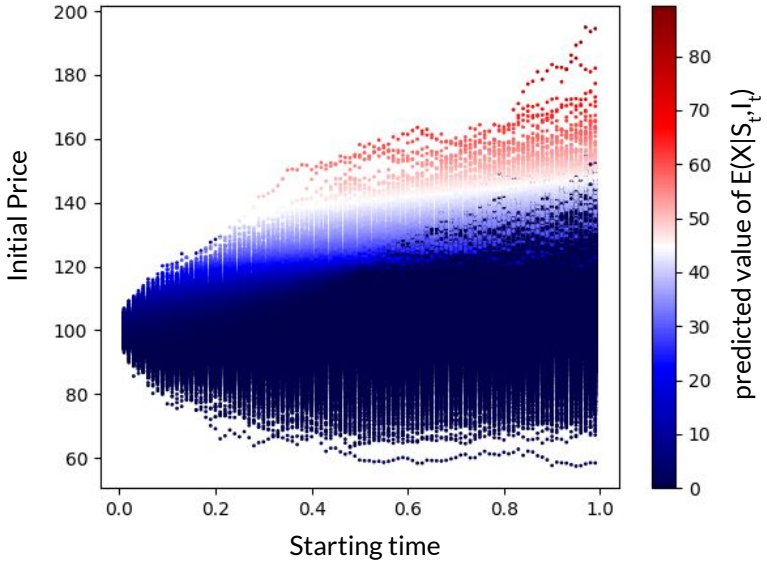


Réseau de neurones

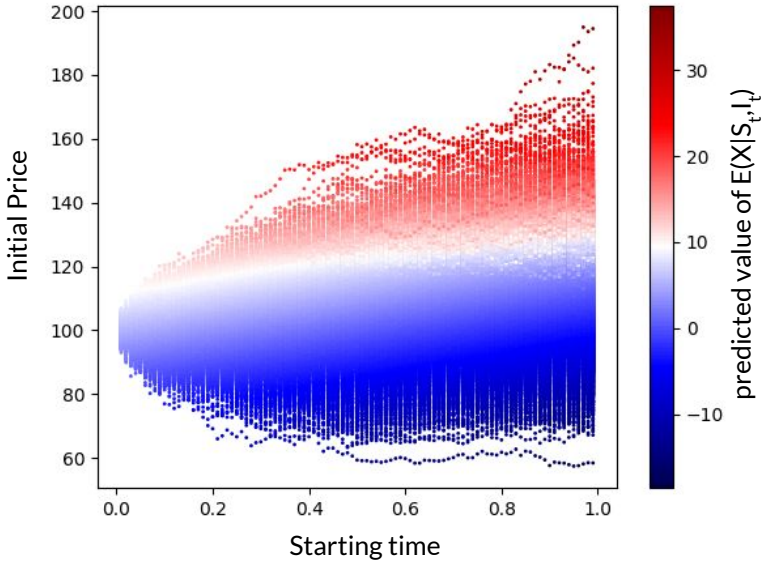


Résultats: Régression linéaire

Monte Carlo imbriqué

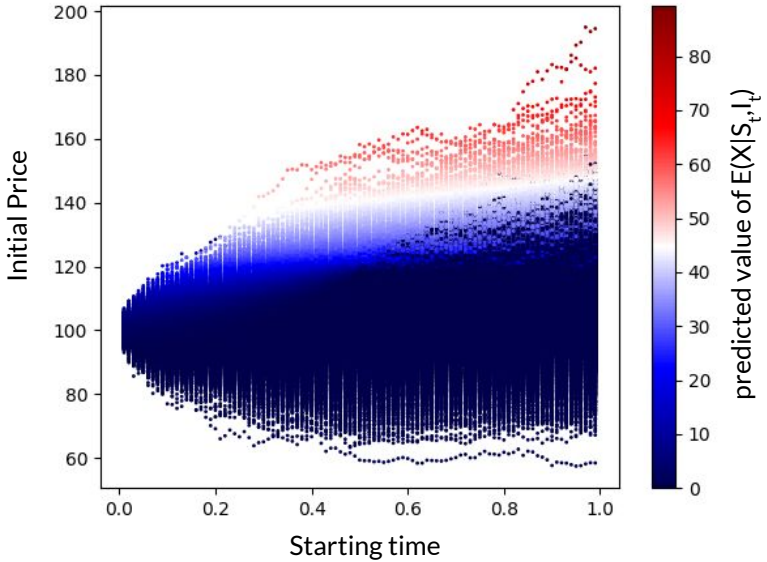


Régression linéaire

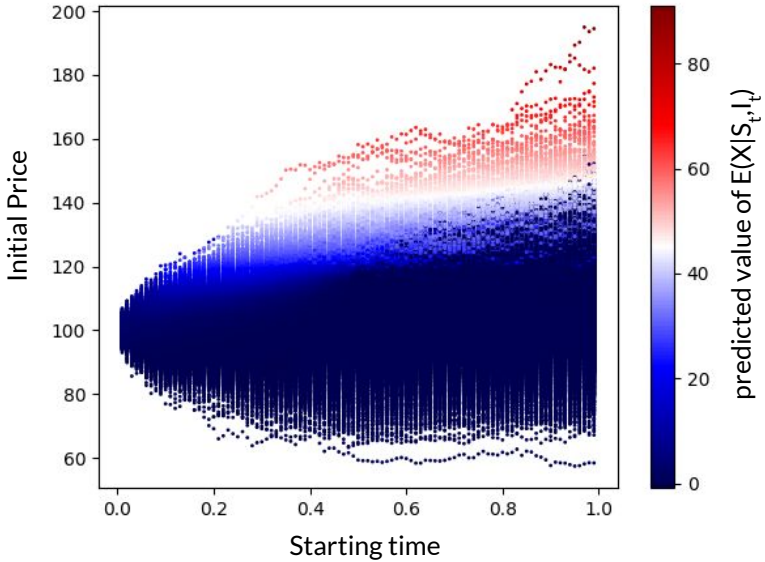


Résultats: Réseau de neurones

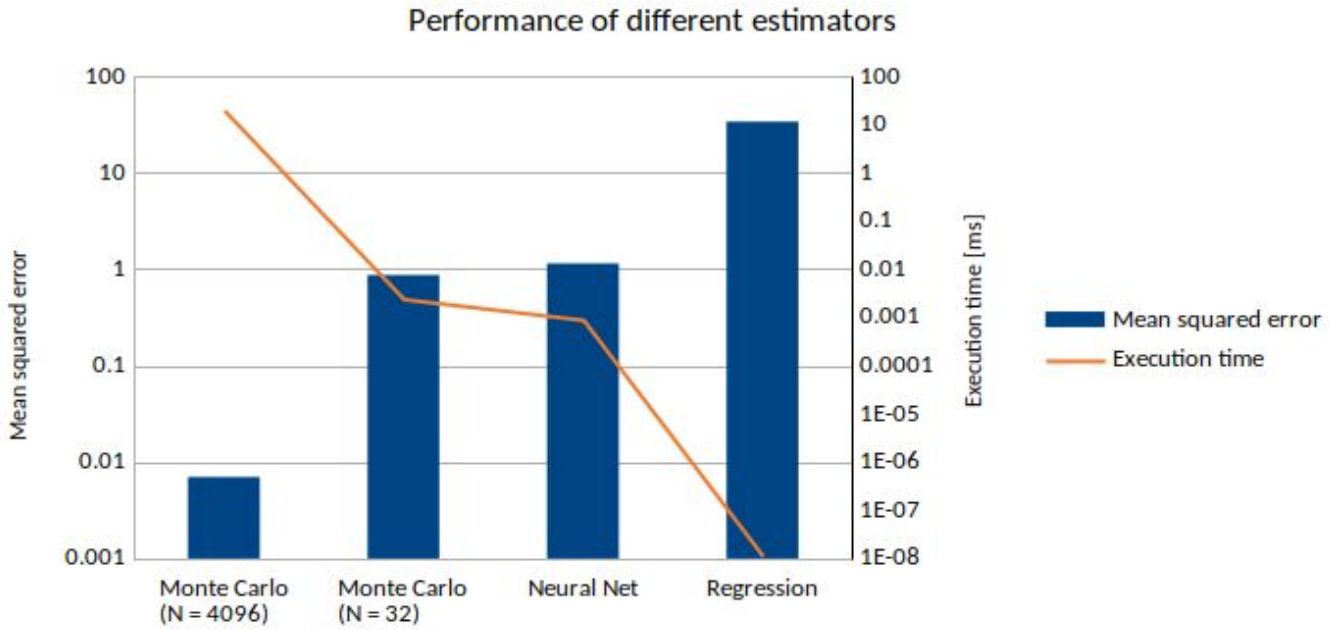
Monte Carlo imbriqué



Réseau de neurones



Comparaison numérique



CODE

```
506
507     // calculate outer trajectories
508     int Nblocks = (Nouter+threads_per_block-1)/threads_per_block;
509
510     MCouter_k<<<Nblocks,threads_per_block>>>
511     (P1, P2, x_0, dt, B, K, leng, M, Nouter, CMRG, time, price, i_t);
512
513     // GPU timer instructions
514     cudaEventCreate(&start);
515     cudaEventCreate(&stop);
516     cudaEventRecord(start,0);
517
518     // calculate inner trajectories
519     Nblocks = (Ninner+threads_per_block-1)/threads_per_block; // ceiling function
520     dim3 dim_blocks(Nouter, Nblocks);
521
522     for(int i = 0; i < M-1; i++){
523         MCinner_k<<<dim_blocks, threads_per_block, 2*threads_per_block*sizeof(float)>>>
524         (P1, P2, dt, B, K, leng, M, Ninner, CMRG, i+1, time+i*Nouter, price+i*Nouter, i_t+i*Nouter, sum+i*Nouter, sum2+i*Nouter);
525     }
526
527     // GPU timer instructions
528     cudaEventRecord(stop,0);
529     cudaEventSynchronize(stop);
530     cudaEventElapsedTime(&Tim,start, stop);
531     cudaEventDestroy(start);
532     cudaEventDestroy(stop);
```

```
__global__ void MCouter_k(int P1, int P2, float x_0, float dt,
                          float B, float K, int L, int M,
                          int Nouter, int Ninner, TabSeedCMRG_t *pt_cmrg,
                          float* time, float* price, int* i_t){

// threadIdx.x and blockIdx.x -> index outer trajectory
int idx_outer = threadIdx.x + blockDim.x * blockIdx.x;
int a0, a1, a2, a3, a4, a5, k, i, q, P;
float g0, g1, Sk, Skp1, t, v;

if(idx_outer < Nouter){

  Sk = x_0;
  P = 0;
  CMRG_get_d(&a0, &a1, &a2, &a3, &a4, &a5, pt_cmrg[0][int(idx_outer/Ninner)][idx_outer%Ninner]);

  for (k=0; k<M-1; k++){
     // calculate stock trajectory
    for (i=1; i<=L; i++){…
    P += (Sk<B);

    // save results
    time[idx_outer+k*Nouter] = t;
    price[idx_outer+k*Nouter] = Sk;
    i_t[idx_outer+k*Nouter] = P;
  }

  CMRG_set_d(&a0, &a1, &a2, &a3, &a4, &a5, pt_cmrg[0][int(idx_outer/Ninner)][idx_outer%Ninner]);
}
```

```
353  __global__ void MCinner_k(int P1, int P2, float dt,
354                            float B, float K, int L, int M,
355                            int Ninner, TabSeedCMRG_t *pt_cmrg, int k_start,
356                            float* time, float* price, int* i_t, float* sum, float* sum2){
357
358    // blockIdx.x -> index outer trajectory
359    int idx_outer = blockIdx.x;
360    // threadIdx.x and blockIdx.y -> index inner trajectory
361    int idx_inner = threadIdx.x + blockDim.x * blockIdx.y;
362
363    int a0, a1, a2, a3, a4, a5, k, i, q, P;
364    float g0, g1, Sk, Skp1, t, v;
365
366    extern __shared__ float H[];
367
```

```
368   if(idx_inner < Ninner){
369
370     Sk = price[idx_outer];
371     P = i_t[idx_outer];
372
373     CMRG_get_d(&a0, &a1, &a2, &a3, &a4, &a5, pt_cmrg[0][idx_outer][idx_inner]);
374
375     // Calculate stock trajectory
376   >  for (k=k_start; k<M; k++){ ···
387     }
388
389     CMRG_set_d(&a0, &a1, &a2, &a3, &a4, &a5, pt_cmrg[0][idx_outer][idx_inner]);
390
391     H[threadIdx.x] = expf(-rt_int(dt*dt*L*k_start, t, 0, q))*fmaxf(0.0f, Sk-K)*((P<=P2)&&(P>=P1))/Ninner;
392     H[threadIdx.x + blockDim.x] = Ninner*H[threadIdx.x]*H[threadIdx.x];
393     __syncthreads();
394
395     i = blockDim.x/2;
396     while (i != 0) {
397       if (threadIdx.x < i){
398         H[threadIdx.x] += H[threadIdx.x + i];
399         H[threadIdx.x + blockDim.x] += H[threadIdx.x + blockDim.x + i];
400       }
401       __syncthreads();
402       i /= 2;
403     }
404     if (threadIdx.x == 0){
405       atomicAdd(sum + idx_outer, H[0]);
406       atomicAdd(sum2 + idx_outer, H[blockDim.x]);
407     }
```

Merci de votre attention.