Implementação da Modelagem Sísmica

Carlos H. S. Barbosa[1] & Luana N. Osório[2]

19 de Outubro de 2018

[1] c.barbosa@nacad.ufrj.br
[2] luana.n.osorio@gmail.com

Sumário

Introdução ao algoritmo de Modelagem Sísmica

1. Geração da fonte sísmica
2. Leitura do arquivo com modelo de velocidade
3. Adição de borda ao modelo de velocidade
4. Loop nos passos de tempo
   - Inserir fonte na matriz do campo inicial
   - Cálculo do campo de pressão no tempo posterior a partir das matrizes dos campos presente e anterior
   - Aplicação da condição de contorno não reflexiva proposta por Reynolds nas bordas
   - Atenuação das amplitudes nas bordas dom modelo, utilizando a técnica proposta por CERJAN
   - Atualização dos campos
   - Cálculo da matriz com registros na posição dos receptores para cada passo de tempo
5. Fim do loop de tempo

Fonte Sísmica

A fonte que empregaremos em nossas simulações será a derivada segunda da função Gaussiana:

$$f(t) = \left[1 - 2\pi(\pi f_c t_d)^2\right] e^{-\pi(\pi f_c t_d)^2}$$

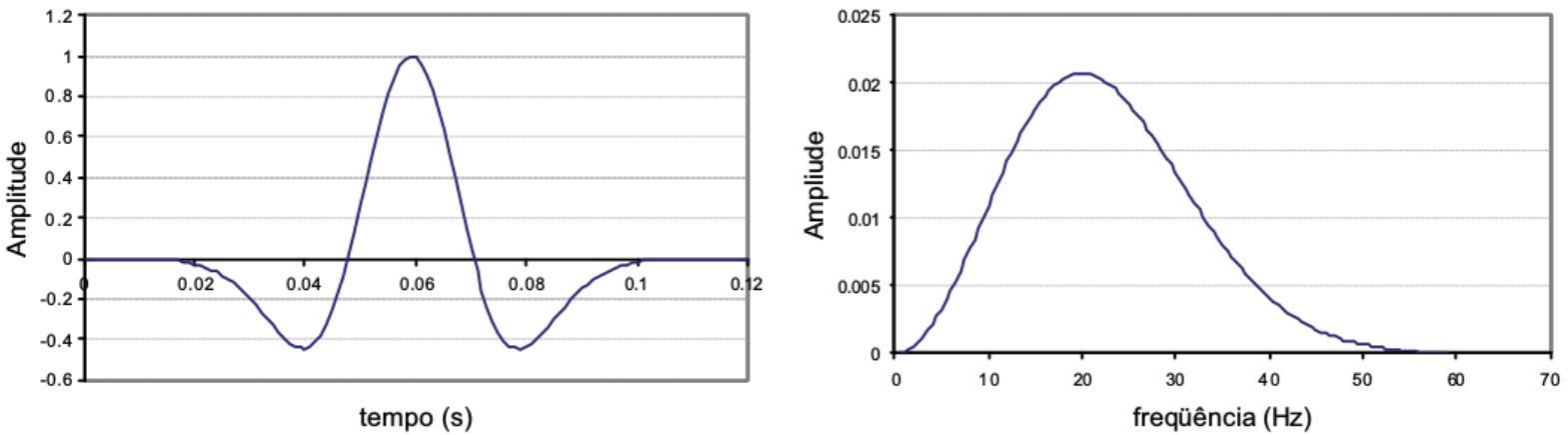
$t \rightarrow$ tempo
$t_d \rightarrow$ tempo defasado
$f_c \rightarrow$ parâmentro relacionado com a frequência de corte da fonte

$$t_d = t - \frac{2\sqrt{\pi}}{f_c} \qquad f_{corte} = 3\sqrt{\pi}\ f_c$$

Fonte Sísmica

Exemplo

Função fonte e sua transformada de Fourier para o caso no qual a frequência de corte é de 60 Hz



Bulcão, A. (2004). Modelagem e Migração Reversa no tempo empregando operadores elásticos. PhD thesis, COPPE/UFRJ.

Fonte Sísmica

```
void ricker_wavelet(float *fonte, int Nf)
{
    float fc;
    float td;
    int time_step;


    fc = FCORTE/(3*sqrt(M_PI));

    for (time_step=0; time_step<Nf; time_step++)
    {
        td = (time_step * DT) - (2* sqrt(M_PI)/FCORTE);

        fonte[time_step]= exp(-M_PI * pow((M_PI*fc*td),2)) * (1. - (2.* M_PI * pow(M_PI*fc*td, 2)));

    }

    printf("\n PARAMETROS DA FONTE: Nf = %d   FCorte = %f   dt = %f \n", Nf, FCORTE, DT);


    FILE * source_File = NULL;
    source_File = fopen ("../output/fonte.bin","wb");
    fwrite (fonte, sizeof(float), Nf, source_File);

}
```

Modelagem Sísmica utilizando os operadores acústicos

$$P^{t+\Delta t}_{[i,j]} = \frac{(\Delta t)^2}{12h^2} c^2_{[i,j]} \left(\begin{array}{c} -P^t_{[i+2,j]} - P^t_{[i-2,j]} - P^t_{[i,j+2]} - P^t_{[i,j-2]} \\ +16(P^t_{[i+1,j]} + P^t_{[i-1,j]} + P^t_{[i,j+1]} + P^t_{[i,j-1]}) \\ -60(P^t_{[i,j]}) \end{array} \right)$$

$$+2P^t_{[i,j]} - P^{t-\Delta t}_{[i,j]}$$

[i, j] $\rightarrow$ índices para referenciar determinado ponto do grid, nas direções x e y, respectivamente.

Δt $\rightarrow$ intervalo de tempo adotado para o avanço da solução ao longo do tempo.

h $\rightarrow$ intervalo espacial entre os pontos da discretização.

P $\rightarrow$ pressão hidrostática do campo de ondas, e os índices sobrescritos e subscritos indicam o instante de tempo considerado e o ponto do grid, respectivamente.

Modelagem Sísmica utilizando os operadores acústicos

```
void wave_equation(float *P0, float *P1, int Nxx, int Nzz, float *vel, float *coefic)
{
    int i, j;

    for (i=2; i<(Nxx-2); i++){
         for (j=2; j<(Nzz-2); j++){

              P1[Nzz*i+j] = coefic[Nzz*i+j]*( P0[Nzz*(i-2)+j] + P0[Nzz*(i+2)+j]+P0[Nzz*i+(j+2)]
              +P0[Nzz*i+(j-2)]-16*(P0[Nzz*(i-1)+j]+P0[Nzz*(i+1)+j]+P0[Nzz*i+(j-1)]+P0[Nzz*i+(j+1)])
              +60*P0[Nzz*i+j])+ 2*P0[Nzz*i+j]-P1[Nzz*i+j];
        }
    }
}
```

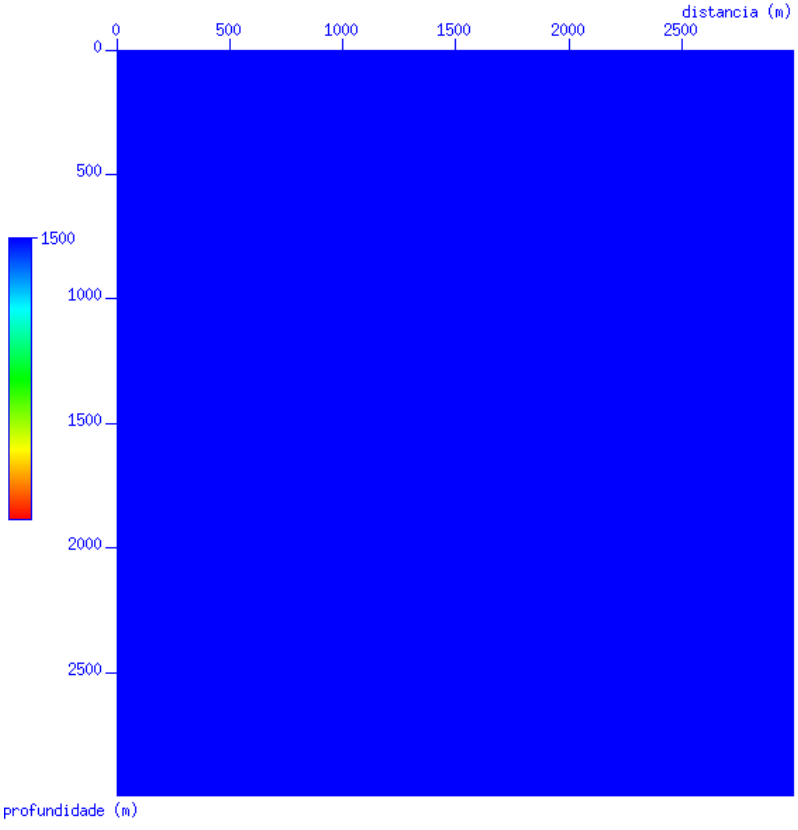
```
void update_wave_field(float *P0, float *P1, int Nxx, int Nzz)
{
    int i, j;
    float p_temp;

    for (i=0; i<Nxx; i++)
    {
        for (j=0; j<Nzz; j++)
        {
            p_temp = P0[Nzz*i + j];
            P0[Nzz*i + j] = P1[Nzz*i + j];
            P1[Nzz*i + j] = p_temp;
        }
    }

}
```

Modelagem Sísmica utilizando os operadores acústicos

```
void modeling_data(int time_step, float *P0, float *P1, float *fonte, float *vel_mod, float *coefic,
float *amp, float *coef_dampz_left, float *coef_dampz_right, float *coef_dampz_inf, float
*coef_dampz_sup, int xshot, int zshot, int Nf, int Nxx, int Nzz, FILE * snapFile_inc){


    if (time_step < Nf){
        P0[Nzz*xshot + zshot] = P0[Nzz*xshot + zshot] + fonte[time_step]*pow((DT/H),
        2)*pow(vel_mod[Nzz*xshot + zshot], 2);
    }

    wave_equation(P0, P1, Nxx, Nzz, vel_mod, coefic);

    if (BORDA_CONTORNO_AMORT == 1){
        apply_Cerjan_Boundary (P0, P1, Nxx, Nzz, coef_dampz_left, coef_dampz_right, coef_dampz_inf,
        coef_dampz_sup);
        apply_Reynolds_Boundary (P0, P1, amp, Nxx, Nzz);
    }

    update_wave_field (P0, P1, Nxx, Nzz);


     if (time_step%100==0){
         fwrite (&P0[0], sizeof(float), Nxx*Nzz, snapFile_inc);
     }

}
```

Experimentos

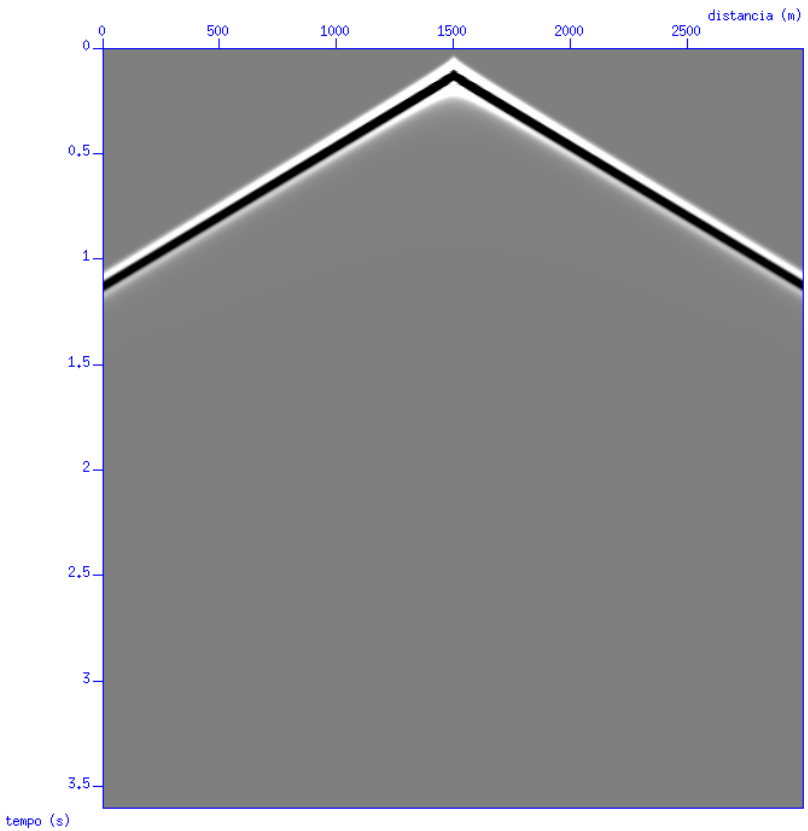
Modelo Homogêneo

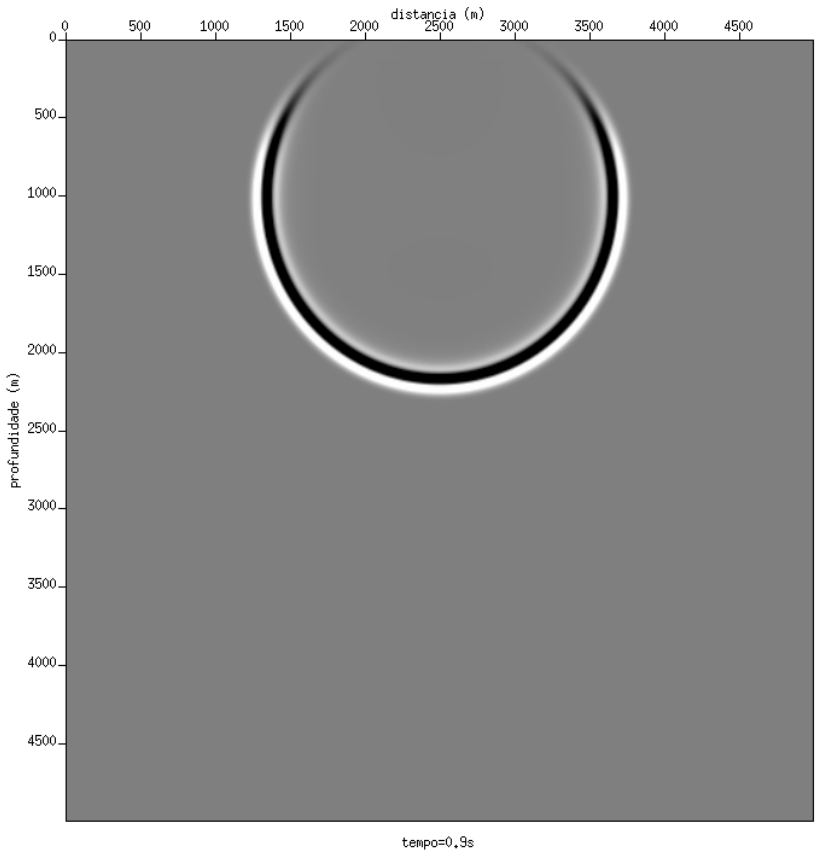


Parâmetros	
H(m)	10
[NX NZ]	[300 300]
NTOTAL	6000
FCORTE (Hz)	30
DT(s)	0.0006
Z0_SHOT	1
X0_SHOT	150
NREC	300

Experimentos

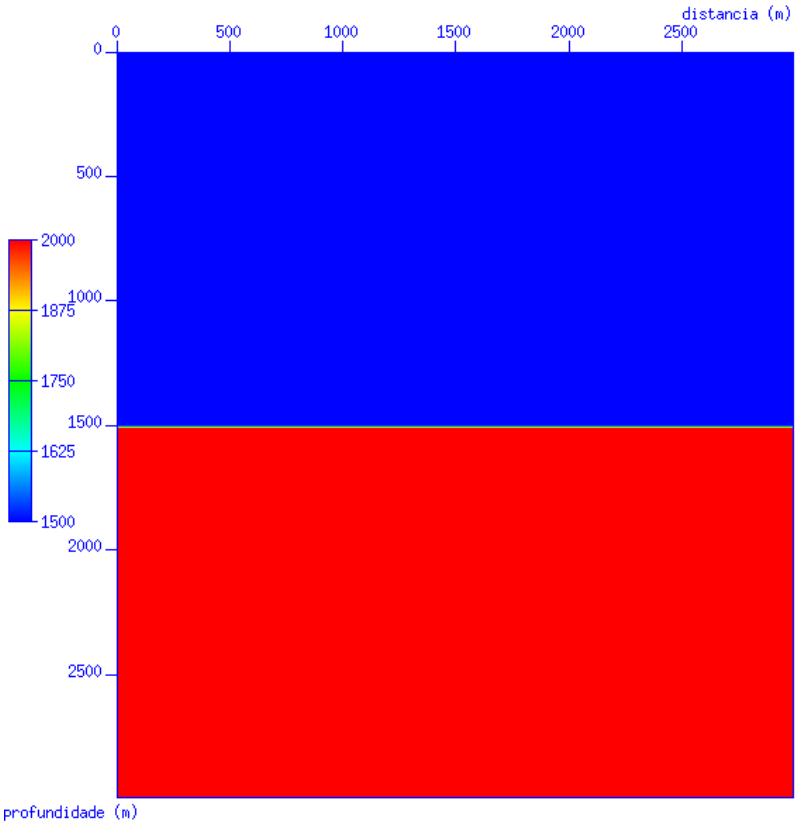
Results





Experimentos

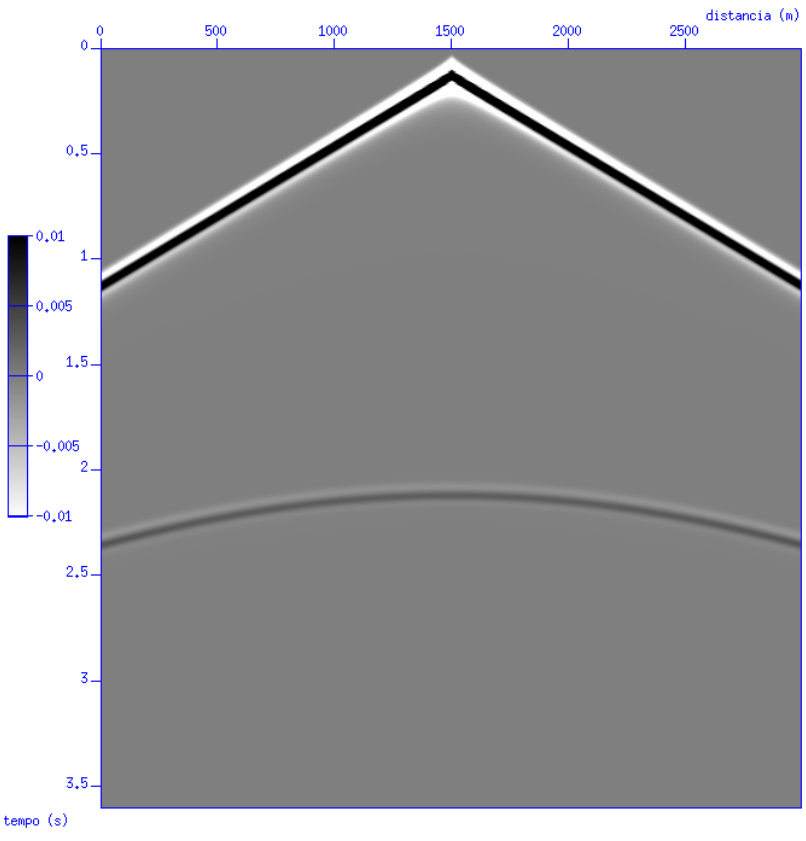
Modelo Plano Paralelo 2 camadas

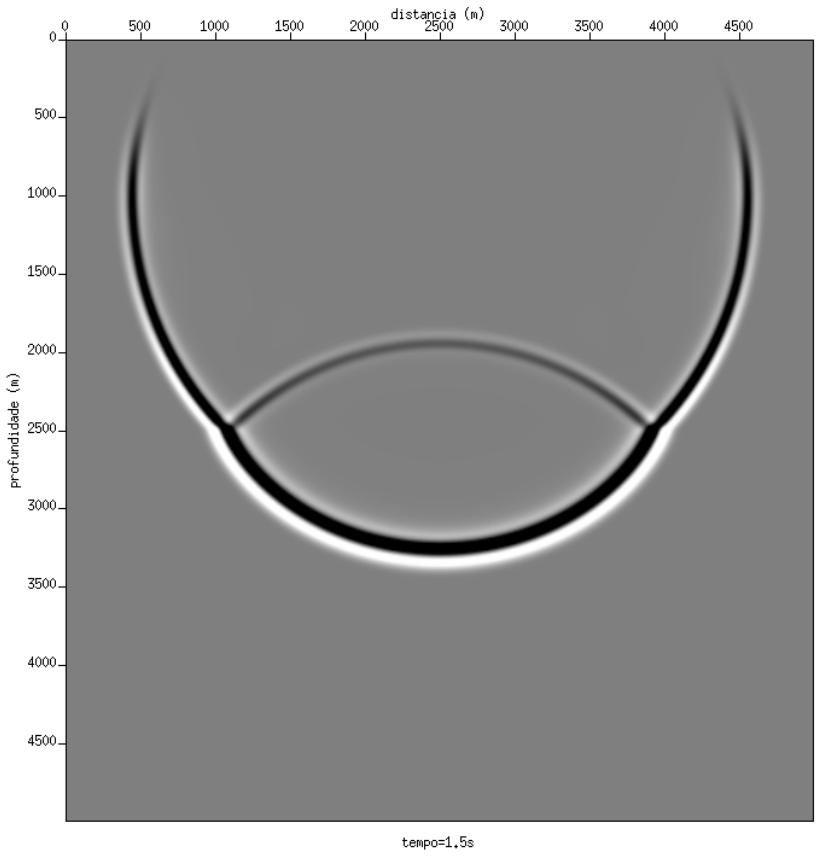


Parâmetros	
H(m)	10
[NX NZ]	[300 300]
NTOTAL	6000
FCORTE (Hz)	30
DT(s)	0.0006
Z0_SHOT	1
X0_SHOT	150
NREC	300

Experimentos

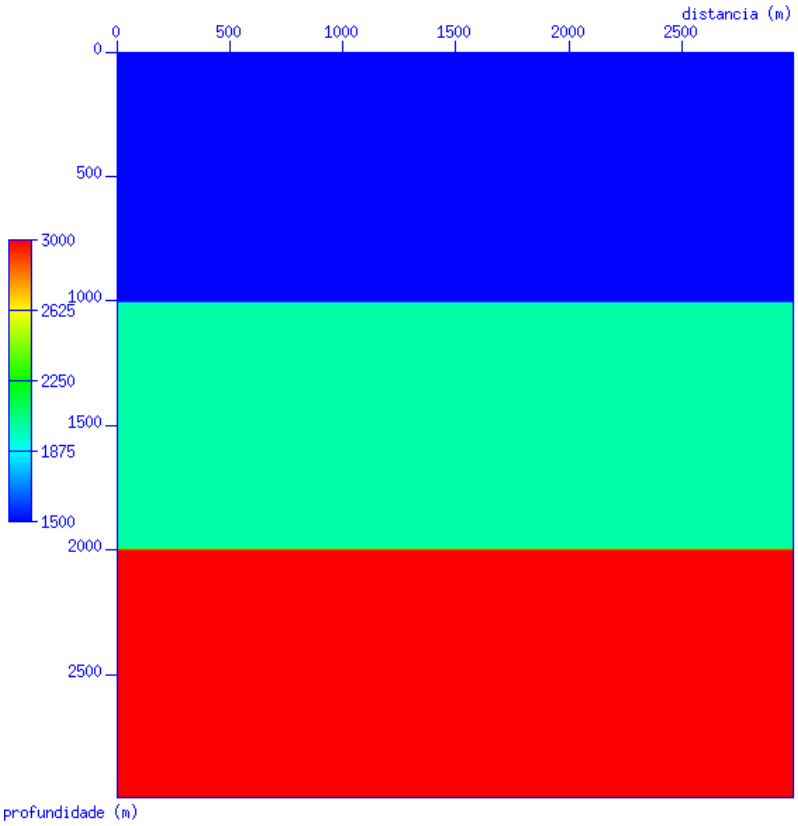
Results





Experimentos

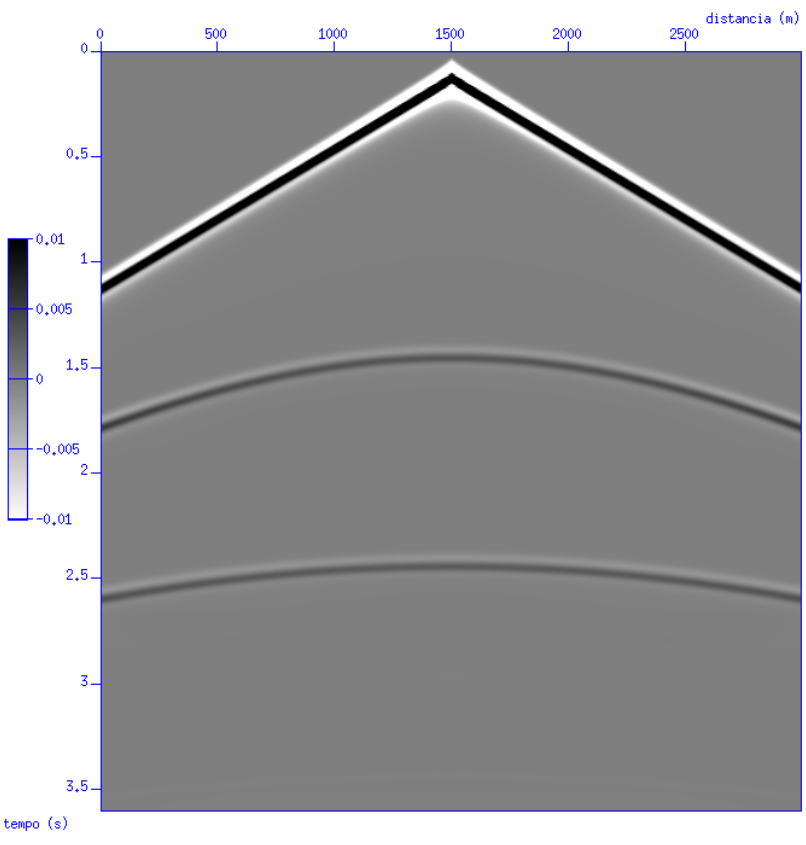
Modelo Plano Paralelo 3 camadas

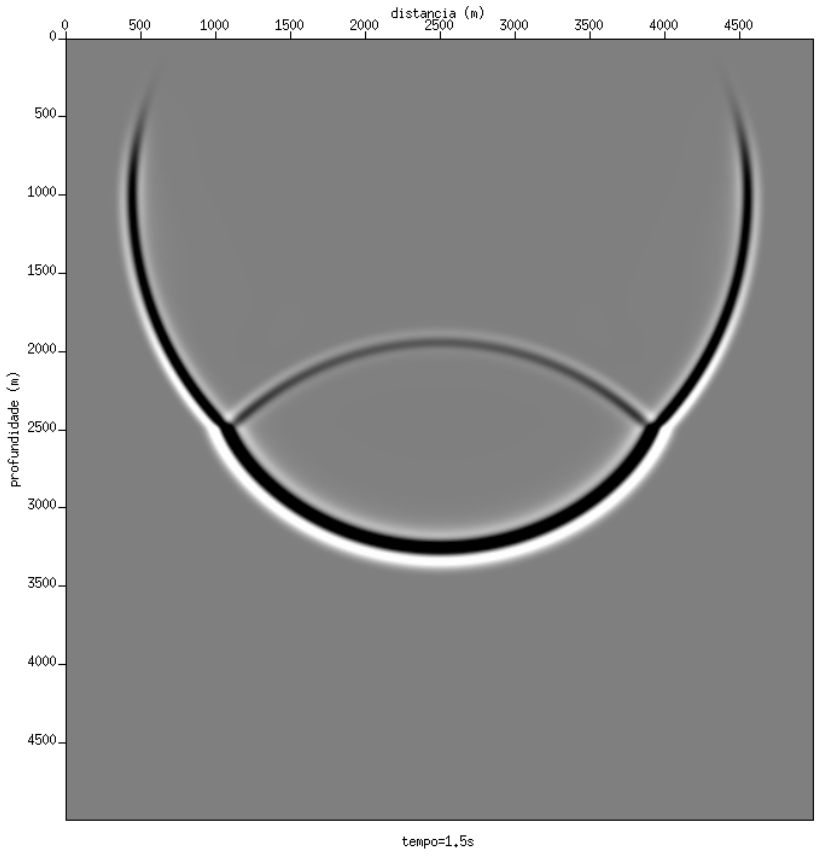


Parâmetros	
H(m)	10
[NX NZ]	[300 300]
NTOTAL	6000
FCORTE (Hz)	30
DT(s)	0.0006
Z0_SHOT	1
X0_SHOT	150
NREC	300

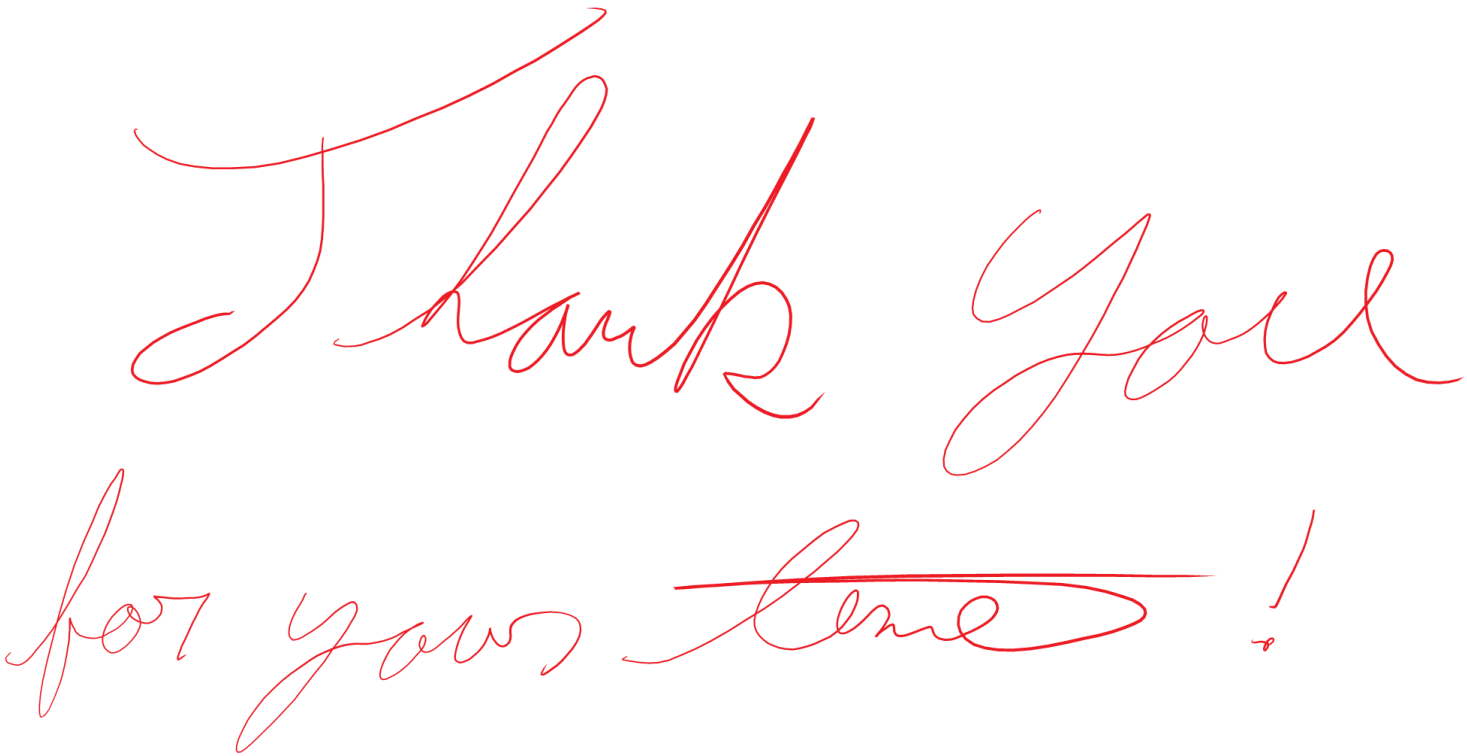
Experimentos

Results





Thank you for your time.



Overleaf Template.