LNCC
www.lncc.br
Laboratório Nacional de Computação Científica

Laboratório Nacional de Computação Científica

GA - 018 - Métodos Numéricos

Exercício de Implementação 3

Aluno: Vinícius Prata Klôh

30 de agosto de 2018

1 Modelo Estudado e Resolução Numérica

O modelo estudado é dado pela Equação 1

$$\begin{cases} \frac{\partial N}{\partial t} = D\frac{\partial^2 x}{\partial x^2} + rN \\ N(t=0,x) = N_0(x) = exp(-200x^2) \ em \ t \in (0,T] \ e \ x \in (-L,L) \\ N(t, x=-L) = N(t, x=L) = 0 \end{cases} \quad (1)$$

onde $\frac{\partial N}{\partial t}$ representa a taxa de variação da densidade (ou população) de células N ao longo do tempo t, D o termo de difusão, $\frac{\partial^2 x}{\partial x^2}$ a taxa de dispersão por difusão no espaço x e r a taxa intrínseca de crescimento. O termos $-L$ e L correspondem aos pontos extremos do intervalo no espaço.

Problema: Determinar a evolução de N ao longo do tempo utilizando o método Euler Implícito e diferenças centradas para resolver numericamente o modelo da Equação 1.

1.1 Resolução Numérica

A Resolução Numérica foi realizada seguindo os passos:

i) Definindo a partição do domínio computacional no tempo, adotando o índice subescrito n, conforme mostra a Figura 1, onde $\Delta t = t_n - t_{n-1}$.

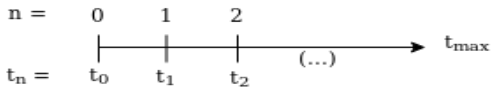


Figura 1: Partição do domínio no tempo.

ii) Definindo a partição do domínio computacional no espaço, adotando o índice sobrescrito i, como apresenta a Figura 2, onde a coordenada de cada ponto foi denotada por x_i e $-x_i$, sendo $i = 0, 1, 2, ..., 200$. Com a partição uniforme, foi definido $\Delta x = \frac{2L}{2Npartes} = x_i - x_{i-i}$, onde N_{partes} corresponde ao número de divisões do domínio espacial.

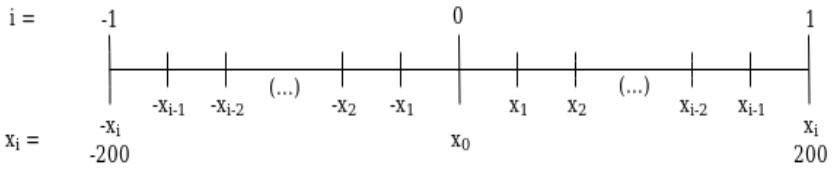


Figura 2: Partição do domínio no espaço.

iii) Aproximando os termos de derivada na Equação 1: $\frac{\partial N}{\partial t} \approx \frac{N_n - N_{n-1}}{\Delta t}$ e $\frac{\partial^2 x}{\partial x^2} \approx \frac{N_n^{i-1} - 2N_n^i + N_n^{i+1}}{\Delta x^2}$.

iv) Substituindo na Equação 1 e manipulando-a algebricamente, foi obtida a Equação 2

$$N_n^i\gamma - \beta(N_n^{i+1} + N_n^{i-1}) = N_{n-1}^i, \forall i = -200, -199, \ldots, 199, 200 \quad (2)$$

onde $\gamma = (1 + 2\beta - \Delta tr)$ e $\beta = \frac{\Delta tD}{\Delta x^2}$.

v) Assim, para cada tempo t_n foi necessário resolver o sistema de equações apresentado à esquerda em (3), para avaliar a Equação 2 em todas as coordenadas do espaço. A Equação 3 apresenta, à direita, o sistema na forma matricial $Ax = b$, sendo A uma matrix tridiagonal simétrica.

$$\begin{cases} N_n^{-200} = 0 \\ N_n^{-199}\gamma - \beta(N_n^{-200} + N_n^{-198}) = N_{n-1}^{-199} \\ N_n^{-198}\gamma - \beta(N_n^{-199} + N_n^{-197}) = N_{n-1}^{-198} \\ \dots \\ N_n^{198}\gamma - \beta(N_n^{199} + N_n^{197}) = N_{n-1}^{198} \\ N_n^{199}\gamma - \beta(N_n^{200} + N_n^{198}) = N_{n-1}^{199} \\ N_n^{200} = 0 \end{cases} \quad \begin{bmatrix} \gamma & -\beta & 0 & \dots & 0 & 0 & 0 \\ -\beta & \gamma & -\beta & \dots & 0 & 0 & 0 \\ 0 & -\beta & \gamma & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \gamma & -\beta & 0 \\ 0 & 0 & 0 & \dots & -\beta & \gamma & -\beta \\ 0 & 0 & 0 & \dots & 0 & -\beta & \gamma \end{bmatrix} \begin{bmatrix} N_n^{-199} \\ N_n^{-198} \\ \\ \dots \\ \\ N_n^{198} \\ N_n^{199} \end{bmatrix} = \begin{bmatrix} N_{n-1}^{-199} \\ N_{n-1}^{-198} \\ \\ \dots \\ \\ N_{n-1}^{198} \\ N_{n-1}^{199} \end{bmatrix} \quad (3)$$

2 Metodologia e Resultados

2.1 Metodologia

Os parâmetros adotados para realização dos experimentos foram: $N_0(x) = exp(-200x^2)$, $\Delta t = 0.1$, $\Delta x = \frac{2}{400} = 5 \times 10^{-3}$, $t_{max} = 1.0$, $r = 0.3$, numéro máximo de iterações $= 20$ e tolerância $= 10^{-5}$. O termo de difusão foi escolhido para dois casos: *a)* No qual as células se difundem mais lentamente pelo espaço, adotando $D = 10^{-5}$ e *b)*, para o qual a difusão ocorre mais rapidamente, com $D = 10^{-2}$. Para a resolução do sistema de equações 3 foi utilizado o método do Gradiente Conjugado com uma matriz solução inicial arbitrária, adotada $x = [0, 0, \dots, 0, 0]^t$. Para cada partição Δx foi repetido o processo iterativo até que o erro relativo $\frac{||(x^{k+1} - x^k)||_\infty}{||(x^{k+1})||_\infty} \leq \delta$, onde δ é a tolerância e k é o índice da iteração.

Foi utilizada também integração númerica simples para estimar uma aproximação para a densidade total de células, onde $I = \int_{n=0.0}^{1.0} \int_{i=-1}^{1} N dt dx \approx \sum_{n=0.0}^{1.0} \sum_{i=-1}^{1} f(N_n^i)$.

No algoritmo para resolvermos computacionalmente o problema foi utilizada a precisão *long double* para evitar erros de arredondamento e necessidade de pré-condicionamento da matriz A em (3).

2.2 Resultados obtidos

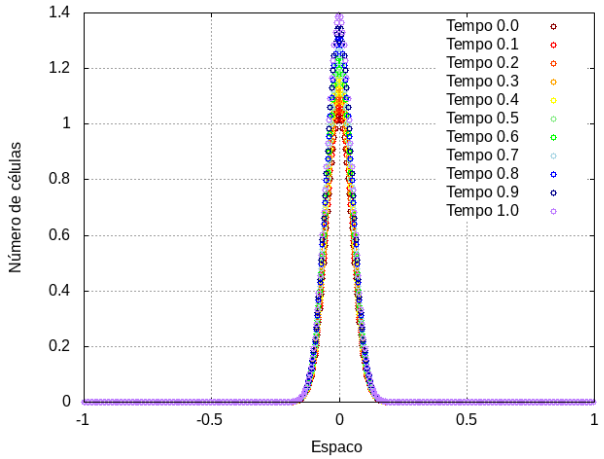
A Figura 3 apresenta os resultados da evolução do número de células no tempo e no espaço para os dois casos estudados *a)* e *b)*. As Figuras 3a, 3c e 3e apresentam o caso *a)*, e as Figuras 3b, 3d e 3f, o caso *b)*. Podemos perceber a influência do termo de difusão na evolução do número de células, quando comparado ao termo reativo. Quando foi adotado o valor do termo difusivo menor do que o valor do termo reativo, a população cresceu ao longo do tempo e manteve-se mais concentrada próxima ao centro do espaço. Quando foi utilizado o termo difusivo com valor maior do que termo reativo, a população se espalhou mais pelo espaço.

Para o caso *a)* a densidade total de células foi de ≈ 332.5 células e para o caso *b)*, foi de ≈ 330.7.

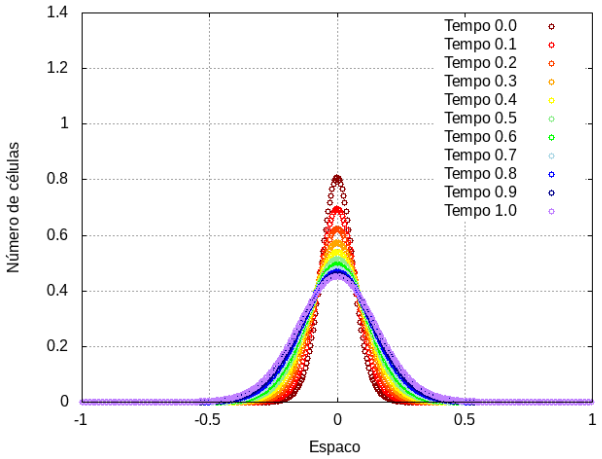
3 Conclusão

Neste trabalho foi estudado o modelo de crescimento exponencial com o termo de dispersão por difusão D. Para resolução numérica do modelo foi utilizado o método de Euler Implícito para aproximação no tempo e método de diferenças centradas para aproximar o operador de derivada segunda, resultando em um sistema esparço. Esse tipo de sistema surge com frequência na resolução de equações com problema de contorno.

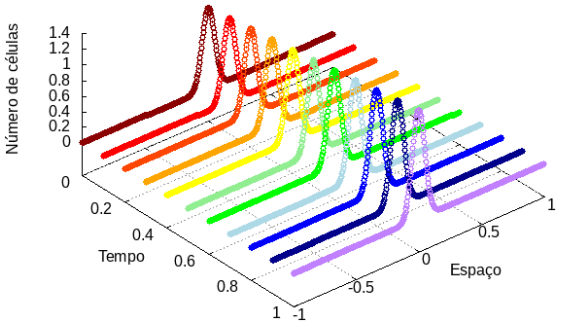
O sistema de equações esparso resultante possui os elementos não nulos seguindo um padrão previsível - matriz tridiagonal neste caso. A matrix A associada ao sistema não precisou ser pré-condicionada, o que proporcionou bons resultados ao empregar o método do Gradiente Conjugado como um método de aproximação iterativa para resolução do sistema.



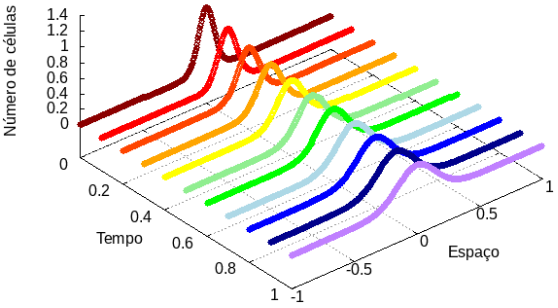
(a) Caso *a)* - Densidade de N no espaço para cada tempo.



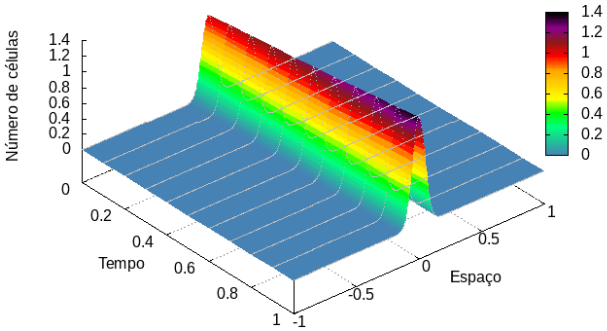
(b) Caso *b)* - Densidade de N no espaço para cada tempo.



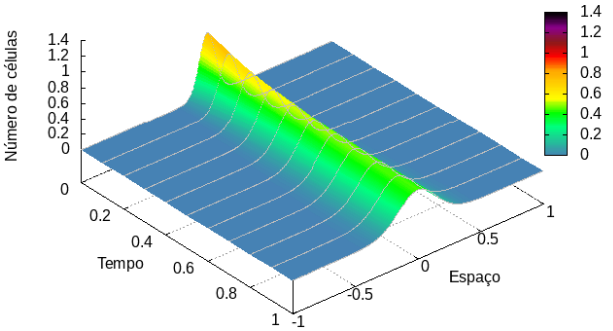
(c) Caso *a)* - Distribuição de N no tempo e no espaço.



(d) Caso *b)* - Distribuição de N no tempo e no espaço.



(e) Caso *a)* - Concentração de N no tempo e no espaço.



(f) Caso *b)* - Concentração de N no tempo e no espaço.

Figura 3: Evolução da população de células.

Questão extra

1 Problema e Resolução Numérica

Problema: Resolver numericamente o modelo da Equação 1 utilizando o método de Euler Explícito para diferentes valores de Δt.

O domínio computacional no tempo foi discretizado conforme a Figura 1, e no espaço, conforme a Figura 2. O termo de segunda ordem também foi aproximado pelo método de diferenças centradas.

Obtivemos então a Equação 4.

$$\frac{N_n^i - N_{n-i}^i}{\Delta t} = D(\frac{N_{n-1}^{i+1} - 2N_{n-1}^i + N_{n-1}^{i-1}}{\Delta x^2}) + rN_{n-i}^i \tag{4}$$

Manipulando-a, chegamos a Equação 5

$$N_n^i = \beta N_{n-1}^{i+1} - \alpha N_{n-1}^i + \beta N_{n-1}^{i-1} \tag{5}$$

onde $\beta = \frac{\Delta t D}{\Delta x^2}$ e $\alpha = 1 + \Delta t r - 2\beta$. Como o modelo foi resolvido de pelo método explícito, os valores no tempo $_{n-1}$ são conhecidos.

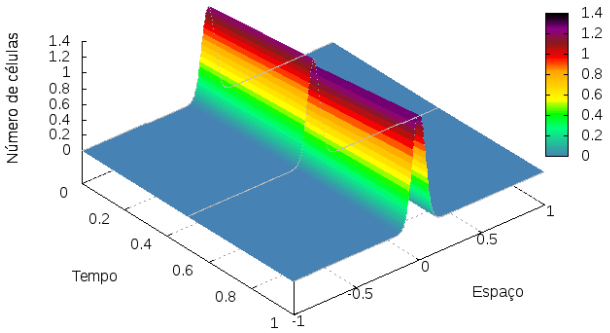
2 Metodologia e Resultados

O caso *a)* foi utilizado para os experimentos adotando diferentes valores para Δt, de forma que o domínio no tempo fosse dividido em partes iguais sem arredondamentos. Os valores de Δt foram então 0.5, 0.2, 0.1, 0.05 e 0.02, permintido avaliar a evolução de N para passos de tempo menores e maiores do que 0.1.

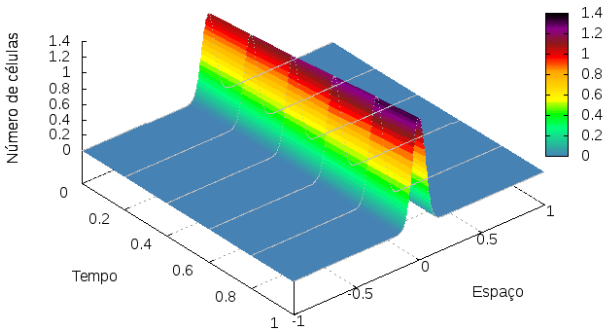
A Figura 4 apresenta a evolução de N no tempo e no espaço para os diferentes valores de Δt. Podemos perceber que, quanto menor foi a partição do domínio no tempo, mais precisas foram as aproximações, devido a construção do método por Série de Taylor. No geral, os resultados foram similares ao resultado apresentado na Figura 3e, onde houve maior concentação de células em torno de x_0 e crescente ao longo do tempo.

3 Conclusão

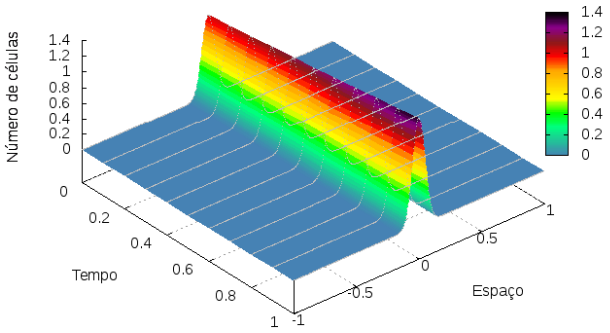
Nesta parte do trabalho foi realizada a resolução numérica do modelo utilizando o método de Euler Explícito. A resolução foi simples se comparada à utilização do método de Euler Implícito, e não foi necessário resolver um sistema linear.



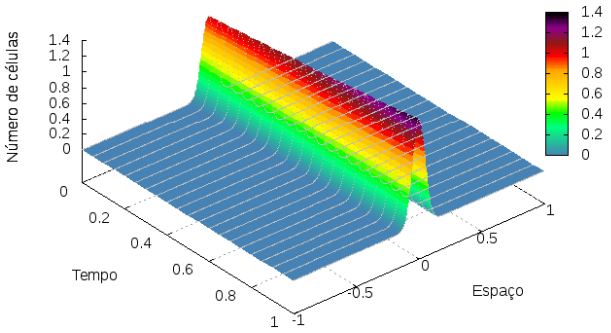
(a) $\Delta t = 0.5$



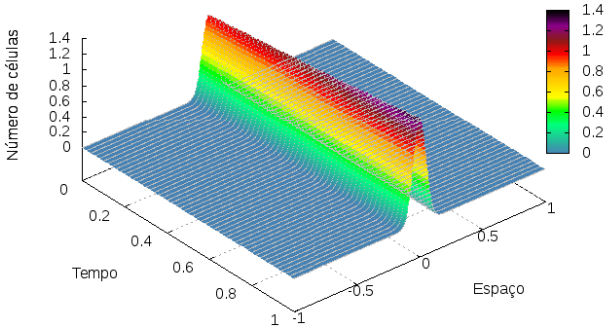
(b) $\Delta t = 0.2$



(c) $\Delta t = 0.1$



(d) $\Delta t = 0.05$



(e) $\Delta t = 0.02$

Figura 4: Evolução da população de células para diferentes passos de tempo.

Apêndice

Neste apêndice são apresentados alguns dos algoritmos construídos para execução computacional dos experimentos. A Figura 4 contêm a função para o método do Gradiente Conjugado e as Figuras 5, 6 e 7, as funções auxiliares para calcular as etapas do método.

Para não aumentar o número de páginas, as demais funções estão disponíveis em `https://github.com/ViniciusPrataKloh/Metodos_numericos_3`, assim como os resultados apresentados.

```c
/*
  Conjugate Gradient function
*/
int cg_method(const Matrix a, const Matrix b, Matrix *x,
              const int iter_max, const long double tol){

  int count = 0;
  int count2 = 0;
  long double error = 0.0;
  Matrix residue_i, residue_ii, direction, xii, lambda, alpha;

  create_matrix(&residue_i, b.row, b.col, 0);     // ix1
  create_matrix(&residue_ii, b.row, b.col, 0);    // ix1
  create_matrix(&direction, b.row, b.col, 0);     // ix1
  create_matrix(&xii, b.row, b.col, 0);           // ix1
  create_matrix(&lambda, 1, 1, 0);                // 1x1
  create_matrix(&alpha, 1, 1, 0);                 // 1x1

  do{
    if(count > 0){  /* For k > 0 */
      count2 = 0.0;
      residuo_calculator(b, a, *x, &residue_ii, count);               /* New residue */
      alpha_calculator(residue_i, residue_ii, &alpha);                /* Alpha */
      direction_calculator(residue_i, residue_ii, alpha, &direction); /* New direction */
      lambda_calculator(residue_ii, direction, a, &lambda, count);    /* Lambda */
      aprox_calculator(*x, lambda, direction, &xii);                  /* Aprox */
    }  else{  /* For k = 0 */
      residuo_calculator(b, a, *x, &residue_i, count);                /* Residue */
      lambda_calculator(residue_i, direction, a, &lambda, count);     /* Lambda */
      aprox_calculator(*x, lambda, residue_i, &xii);                  /* Aprox */
    }
    if(!check_error(*x, xii, tol)){   /* check error */
      for(count2 = 0; count2 <= xii.row; count2++){
        x[0].rect[count2] = xii.rect[count2];
      }
      count++;
      return 0;
    }
    for(count2 = 0; count2 <= xii.row; count2++){
      x[0].rect[count2] = xii.rect[count2];
    }
    count++;
  } while(count <= iter_max);

  return 0;
}
```

Figura 5: Algoritmo para o método do Gradiente Conjugado.

```c
/*
  auxiliar functions
*/
int residuo_calculator(const Matrix b, const Matrix a, const Matrix x, Matrix *residue,
                       const int k){
  int count = 0;

  Matrix ax;
  create_matrix(&ax, b.row, b.col, 0);  // ix1
  mul_matrix(a, x, &ax);
  sub_matrix(b, ax, residue);

  free(ax.rect);
  return 0;
}

int direction_calculator(const Matrix residue_i, const Matrix residue_ii,
                         const Matrix alpha, Matrix *direction){

  int count = 0;
  Matrix alpha_p;
  create_matrix(&alpha_p, residue_i.row, residue_i.col, 0);  // ix1
  mul_matrix(alpha, residue_i, &alpha_p);
  sum_matrix(residue_ii, alpha_p, direction);

  free(alpha_p.rect);
  return 0;
}
```

Figura 6: Funções auxiliares para o cálculo do resíduo e direção de correção.

```c
int lambda_calculator(const Matrix residue, const Matrix direction,
                      const Matrix a, Matrix *lambda, const int k){

  Matrix rr, ar, rar;
  create_matrix(&rr, 1, 1, 0);                        // 1x1
  create_matrix(&ar, residue.row, residue.col, 0);    // ix1
  create_matrix(&rar, 1, 1, 0);                       // 1x1

  if(k > 0){
    Matrix rr_ii;
    create_matrix(&rr_ii, 1, 1, 0);                   // 1x1
    mul_matrix(residue, residue, &rr);
    mul_matrix(a, direction, &ar);
    mul_matrix(direction, ar, &rar);
    free(rr_ii.rect);
  } else{
      mul_matrix(residue, residue, &rr);
      mul_matrix(a, residue, &ar);
      mul_matrix(residue, ar, &rar);
  }
  lambda[0].rect[0] = rr.rect[0] / rar.rect[0];

  free(rr.rect);
  free(ar.rect);
  free(rar.rect);
  return 0;
}

int alpha_calculator(const Matrix residue_i, const Matrix residue_ii, Matrix *alpha){

  int count = 0.0;
  Matrix rr_i, rr_ii;
  create_matrix(&rr_i, 1, 1, 0);     // 1x1
  create_matrix(&rr_ii, 1, 1, 0);    // 1x1

  mul_matrix(residue_i, residue_i, &rr_i);
  mul_matrix(residue_ii, residue_ii, &rr_ii);

  alpha[0].rect[0] = (rr_ii.rect[0] / rr_i.rect[0]);

  free(rr_i.rect);
  free(rr_ii.rect);
  return 0;
}
```

Figura 7: Funções auxiliares para o cálculo de λ e α.

```c
int aprox_calculator(const Matrix xi, const Matrix lambda, const Matrix residue,
                     Matrix *xii){

  Matrix lambda_r, aux;
  int count;
  create_matrix(&lambda_r, residue.row, residue.col, 0);  // 4x1
  create_matrix(&aux, residue.row, residue.col, 0);       // 4x1
  mul_matrix(lambda, residue, &lambda_r);
  sum_matrix(xi, lambda_r, &aux);

  for(count = 0; count < aux.row; count++){
    xii[0].rect[count] = aux.rect[count];
  }

  free(lambda_r.rect);
  free(aux.rect);
  return 0;
}

int check_error(const Matrix xi, const Matrix xii, const long double tol){

  int count = 0;
  long double error = 0.0, max = 0.0;
  for(count = 0; count < xi.row; count++){
    error = (xii.rect[count] - xi.rect[count]) / xi.rect[count];
    if(error > max)
      max = error;
  }
  if(max >= tol)
    return 1;
  return 0;
}
/*
  end auxiliar functions
*/
```

Figura 8: Funções auxiliares para o cálculo da aproximação para x e verificação dos erros a cada iteração.