A Fast Solution Method for Space Fractional Diffusion Equations

Gregory Capra

Occidental College

April 7, 2016

Outline

- What is diffusion?
- Fractional Calculus
- The Discretization
- The Crank-Nicolson Method
- Gaussian Elimination & Matrix Inversion
- Convergence, Memory, & Time Complexity
- The Richardson Extrapolation
- The Conjugate Gradient Squared (CGS) Algorithm
- The Fast Method
- Concluding remarks

Standard Diffusion

You should be familiar with these:

- Food coloring in water
- cigarette smoke into the air
- water into cooking noodles (pasta)

Standard Diffusion

You should be familiar with these:

- Food coloring in water
- cigarette smoke into the air
- water into cooking noodles (pasta)

and maybe these:

- oxygen from plant cells into the air
- oxygen from blood cells in the human body's blood stream into muscles

Everything isn't Standard

Standard Diffusion $\rightarrow$ random motion & "mean-free path" $\rightarrow$ Gaussian Distribution

Everything isn't Standard

Standard Diffusion $\rightarrow$ random motion & "mean-free path" $\rightarrow$ Gaussian Distribution

$$\sigma_r^2 \sim Dt^1 \quad \rightarrow \quad \frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}$$

Everything isn't Standard

Standard Diffusion $\rightarrow$ random motion & "mean-free path" $\rightarrow$ Gaussian Distribution

$$\sigma_r^2 \sim Dt^1 \quad \rightarrow \quad \frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}$$

Anomalous Diffusion $\rightarrow$ inter-dependencies & random fluctuations $\rightarrow$ Pareto Distribution

Everything isn't Standard

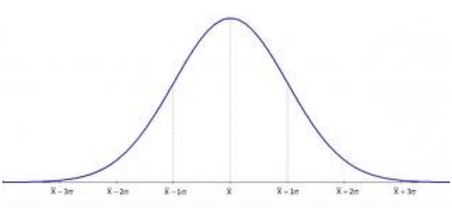
Standard Diffusion → random motion & "mean-free path" → Gaussian Distribution

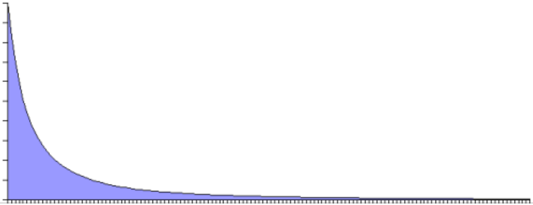
$$\sigma_r^2 \sim Dt^1 \quad \rightarrow \quad \frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}$$

Anomalous Diffusion → inter-dependencies & random fluctuations → Pareto Distribution

$$\sigma_r^2 \sim Dt^\alpha \quad \rightarrow \quad \frac{\partial u}{\partial t} = D\frac{\partial^\alpha u}{\partial x^\alpha}$$

Gaussian vs. Pareto Distribution





Anomalous Diffusion

Definition: Anomalous Diffusion is a diffusion process that has a non-linear relationship to time, i.e., it is affected by random fluctuations and dependencies.[1]

Examples include:

- Transport of electrons in a photocopier
- Foraging behavior of animals
- Trapping of contaminants in groundwater
- Proteins across cell membranes

Super-Diffusion: Diffusion which exceeds linear relationship with time, faster than classical diffusion. $Dt^{\alpha}, \alpha > 1$

Sub-Diffusion: Diffusion less than linear, slower than classical diffusion. $Dt^{\alpha}, \alpha < 1$ [1]

Calculus Anybody?

The basic definition of the whole derivative of a function $f(x)$:

$$\frac{d}{dx}f(x) = \lim_{h\to 0}\frac{f(x)-f(x-h)}{h}$$

Repeated composition of this operation leads to

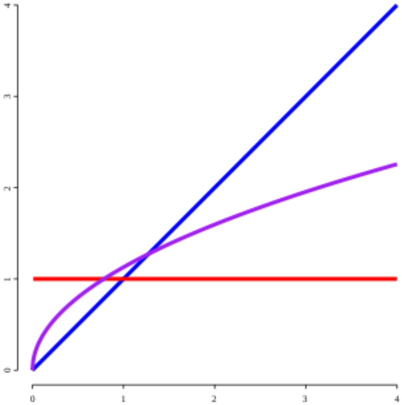
$$\frac{d^n}{dx^n}f(x) = \lim_{h\to 0}\frac{1}{h^n}\sum_{j=0}^{n}(-1)^j\binom{n}{j}f(x-jh), \forall n \in \mathbb{Z}^+$$

Grünwald-Letnikov fractional p-th order derivative of $f(x)$ [2]:

$$\frac{d^p}{dx^p}f(x) = \lim_{h\to 0}\sum_{j=0}^{\left\lfloor\frac{x-x_0}{h}\right\rfloor}\frac{\Gamma(j-p)}{\Gamma(-p)\Gamma(j+1)}f(x-jh)$$

$$p \in R^+$$

Half-Derivative of x



Blue: $f(x) = x$ Red: $f'(x) = 1$ Purple: $f^{(\frac{1}{2})}(x) = 2\sqrt{\dfrac{x}{\pi}}$

$$\int x^n dx = \frac{x^{n+1}}{n+1} + C \quad n \neq -1$$

$$\int x^n dx = \frac{x^{n+1}}{n+1} + C \quad n \neq -1$$

The result of integrating a function is non-local; it depends on how the function behaves over the range for which the integration is performed, not just at a single point.

$$\int x^n dx = \frac{x^{n+1}}{n+1} + C \quad n \neq -1$$

The result of integrating a function is non-local; it depends on how the function behaves over the range for which the integration is performed, not just at a single point.

$$\frac{d}{dx} x^n = n x^{n-1}$$

$$\int x^n dx = \frac{x^{n+1}}{n+1} + C \qquad n \neq -1$$

The result of integrating a function is non-local; it depends on how the function behaves over the range for which the integration is performed, not just at a single point.

$$\frac{d}{dx} x^n = n x^{n-1}$$

However, differentiation is thought of as local, because whole derivatives happen to possess this attribute.

$$\int x^n dx = \frac{x^{n+1}}{n+1} + C \qquad n \neq -1$$

The result of integrating a function is non-local; it depends on how the function behaves over the range for which the integration is performed, not just at a single point.

$$\frac{d}{dx} x^n = n x^{n-1}$$

However, differentiation is thought of as local, because whole derivatives happen to possess this attribute.

The apparent paradoxes of fractional derivatives arise from the fact that, in general, differentiation is non-local, just as in integration!

Where Do We Go From Here

Anomalous diffusion is modeled with Partial Differential Equations (PDE's) that incorporate these fractional derivatives.

Where Do We Go From Here

Anomalous diffusion is modeled with Partial Differential Equations (PDE's) that incorporate these fractional derivatives.

Non-local

Where Do We Go From Here

Anomalous diffusion is modeled with Partial Differential Equations (PDE's) that incorporate these fractional derivatives.

Non-local $\rightarrow$ knowledge of previous solutions

Where Do We Go From Here

Anomalous diffusion is modeled with Partial Differential Equations (PDE's) that incorporate these fractional derivatives.

Non-local $\rightarrow$ knowledge of previous solutions $\rightarrow$ need to store these solutions

Where Do We Go From Here

Anomalous diffusion is modeled with Partial Differential Equations (PDE's) that incorporate these fractional derivatives.

Non-local $\rightarrow$ knowledge of previous solutions $\rightarrow$ need to store these solutions $\rightarrow$ full matrices

Where Do We Go From Here

Anomalous diffusion is modeled with Partial Differential Equations (PDE's) that incorporate these fractional derivatives.

Non-local $\rightarrow$ knowledge of previous solutions $\rightarrow$ need to store these solutions $\rightarrow$ full matrices

- Most problems cannot be solved directly, instead you have to approximate the solution

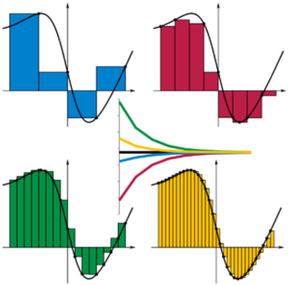
Discretization

Often the framework for setting up an approximate solution is with a **discretization**.

Discretization

Often the framework for setting up an approximate solution is with a **discretization**.

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{\infty} f(x_i^*)\Delta x$$



The Typical Initial-Boundary Value Problem (IBVP)

The two-sided diffusion equation, $1 < \alpha < 2$:

$$\frac{\partial u(x,t)}{\partial t} - d_+(x,t)\frac{\partial^\alpha u(x,t)}{\partial_+ x^\alpha} - d_-(x,t)\frac{\partial^\alpha u(x,t)}{\partial_- x^\alpha} = f(x,t).$$

Domain:

$$x_L \le x \le x_R, \quad 0 < t \le T,$$

Boundary Conditions:

$$u(x_L,t) = 0, \quad u(x_R,t) = 0.$$

Initial Condition:

$$u(x,0) = u_0(x)$$

IBVP cont.

Discretization [10]:

- step size h: $h = \frac{(x_R - x_L)}{N}$
- time step Δt: $\Delta t = \frac{T}{M}$
- spatial partition: $x_i = x_L + ih$ for $i = 0, 1, \ldots, N$
- temporal partition: $t^m = m\Delta t$ for $m = 0, 1, \ldots, M$
- notation: $u_i^m = u(x_i, t^m), d_{+,i}^m = d_+(x_i, t^m), d_{-,i}^m = d_-(x_i, t^m)$, and $f_i^m = f(x_i, t^m)$.

"Simplifying" the Equation

To replace the fractional derivatives we use the Grünwald approximations, shown earlier.

The resulting equation is such:

$$\frac{\partial u(x,t)}{\partial t} - \frac{d_+(x,t)}{h^\alpha}\left(\sum_{k=0}^{i+1} g_k^{(\alpha)} u_{i-k+1}^m + O(h)\right) - \frac{d_-(x,t)}{h^\alpha}\left(\sum_{k=0}^{N-i+1} g_k^{(\alpha)} u_{i+k-1}^m + O(h)\right) = f(x,t).$$

The Grünwald weights g_k^α are defined with $g_k^\alpha = (-1)^k \binom{\alpha}{k}$ where $\binom{\alpha}{k}$ are binomial coefficients of order α. These weights satisfy the following recursive relation [14]:

$$g_0^{(\alpha)} = 1, \quad g_k^{(\alpha)} = \left(1 - \frac{\alpha+1}{k}\right) g_{k-1}^{(\alpha)} \quad \text{for} \quad k \geq 1.$$

The Crank-Nicolson (CN) Method

The general diffusion equation:

The Crank-Nicolson (CN) Method

The general diffusion equation:

$$\frac{\partial u}{\partial t} = F\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^\alpha u}{\partial x^\alpha}\right)$$

The finite difference scheme [11]:

The Crank-Nicolson (CN) Method

The general diffusion equation:

$$\frac{\partial u}{\partial t} = F\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^\alpha u}{\partial x^\alpha}\right)$$

The finite difference scheme [11]:

$$\frac{u_i^{m+1} - u_i^m}{\Delta t} = \frac{1}{2}\left[F_i^{m+1}\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^\alpha u}{\partial x^\alpha}\right) + F_i^m\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^\alpha u}{\partial x^\alpha}\right)\right]$$

The Crank-Nicolson (CN) Method

The general diffusion equation:

$$\frac{\partial u}{\partial t} = F\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^\alpha u}{\partial x^\alpha}\right)$$

The finite difference scheme [11]:

$$\frac{u_i^{m+1} - u_i^m}{\Delta t} = \frac{1}{2}\left[F_i^{m+1}\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^\alpha u}{\partial x^\alpha}\right) + F_i^m\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^\alpha u}{\partial x^\alpha}\right)\right]$$

The purpose of the finite difference method is to approximate the solution to a differential equation by approximating the derivatives with "finite differences".

The Crank-Nicolson (CN) Method

The general diffusion equation:

$$\frac{\partial u}{\partial t} = F\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^\alpha u}{\partial x^\alpha}\right)$$

The finite difference scheme [11]:

$$\frac{u_i^{m+1} - u_i^m}{\Delta t} = \frac{1}{2}\left[F_i^{m+1}\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^\alpha u}{\partial x^\alpha}\right) + F_i^m\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^\alpha u}{\partial x^\alpha}\right)\right]$$

The purpose of the finite difference method is to approximate the solution to a differential equation by approximating the derivatives with "finite differences".

This method is second-order in time, i.e., $O((\Delta t)^2)$. It is first-order in space, i.e., $O((\Delta x))$. These are **convergence rates**. [11]

It was stated earlier that the Crank-Nicolson converges at a rate of $O((\Delta t)^2) + O(\Delta x)$.

It was stated earlier that the Crank-Nicolson converges at a rate of $O((\Delta t)^2) + O(\Delta x)$.

Convergence is dragged down by $O(\Delta x)$. It will follow linear rate.

It was stated earlier that the Crank-Nicolson converges at a rate of $O((\Delta t)^2) + O(\Delta x)$.

Convergence is dragged down by $O(\Delta x)$. It will follow linear rate.

In order to improve to second-order in space, we will use the well-known **Richardson Extrapolation** [11]

So $O(\Delta x) \to O((\Delta x)^2)$

How it Works

The Extrapolation: $2u_i^{m+1}\left(\frac{h}{2}\right) - u_i^{m+1}(h) \Rightarrow u_i^{m+1}(h)$

How it Works

$$\text{The Extrapolation: } 2u_i^{m+1}\left(\frac{h}{2}\right) - u_i^{m+1}(h) \Rightarrow u_i^{m+1}(h)$$

for example, the regular solution $(N = 5)$ and twice-refined $(N = 10)$:

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} \qquad \begin{pmatrix} v_1 \\ \color{red}{v_2} \\ v_3 \\ \color{red}{v_4} \\ v_5 \\ \color{red}{v_6} \\ v_7 \\ \color{red}{v_8} \\ v_9 \end{pmatrix}$$

Test Problem

Fractional Order: $\alpha = 1.8$

Spatial Domain: $x_L = 0$, $x_R = 1$

Time Interval: $t_L = 0$, $t_R = 1$

d_+ coefficient: $.000264\gamma(1.2)(x^\alpha)$

d_- coefficient: $.000264\gamma(1.2)((1-x)^\alpha)$

Source term:

$$f(x,t) = -.0032e^{-t}\Bigg(5000x^2(1-x)^2 + 2.64\left(x^2 + (1-x)^2\right) - 13.2\left(x^3 + (1-x)^3\right) + 12\left(x^4 + (1-x)^4\right)\Bigg)$$

Initial Condition: $f(x) = 16x^2(1-x)^2$

True Solution: $f(x,t) = 16e^{-t}x^2(1-x)^2$

Measuring Error

Since we know the true solution, we can generate an error vector by taking the difference between the true solution vector and our generated approximation.

The two most common ways to analyze that error vector is with the L^2 or L^∞ norm, defined below:

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

$$\|x\|_\infty = \max_{i=1,\ldots,n}|x_i|$$

Non-Extrapolated Results

Without the Extrapolation:

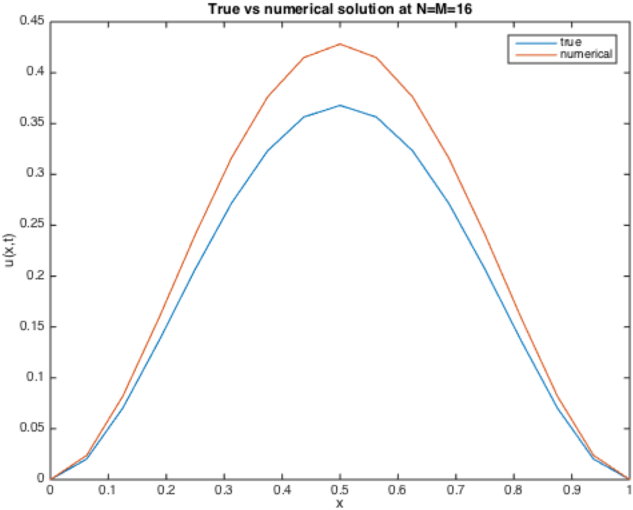
N=M	$\|x\|_\infty$	error ratio
2^2	2.19×10^{-1}	
2^3	1.17×10^{-1}	1.8758
2^4	6.03×10^{-2}	1.9347
2^5	3.07×10^{-2}	1.9665
2^6	1.55×10^{-2}	1.9830
2^7	7.77×10^{-3}	1.9915
2^8	3.89×10^{-3}	1.9957
2^9	1.95×10^{-3}	1.9979
2^{10}	9.74×10^{-4}	1.9989

Extrapolated Results

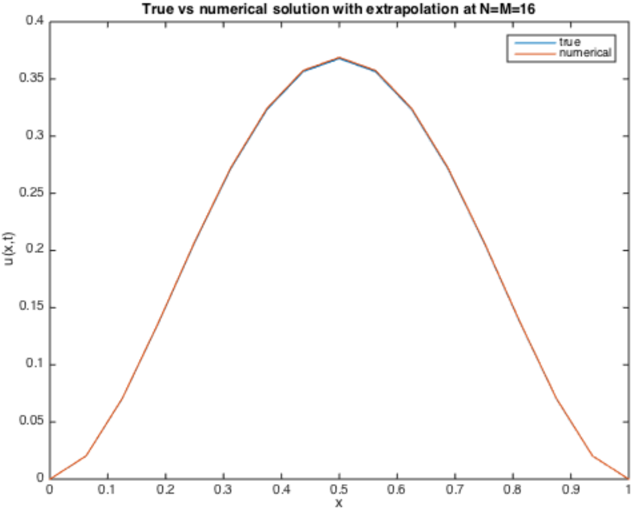
With the Extrapolation:

N=M	$\|x\|_\infty$	error ratio
2^2	1.45×10^{-2}	
2^3	3.94×10^{-3}	3.6802
2^4	1.03×10^{-3}	3.8349
2^5	2.62×10^{-4}	3.9161
2^6	6.62×10^{-5}	3.9578
2^7	1.66×10^{-5}	3.9788
2^8	4.17×10^{-6}	3.9894
2^9	1.04×10^{-6}	3.9947

Non-Extrapolated Graph



Extrapolated Graph



Crank-Nicolson cont.

When applying Crank-Nicolson to our PDE from earlier, the resulting equation can be expressed in matrix form as such:

$$\left(I + \frac{\Delta t}{2h^{\alpha}} A^{m+1}\right) u^{m+1} = \left(I - \frac{\Delta t}{2h^{\alpha}} A^{m}\right) u^{m} + \frac{\Delta t}{2}\left(f^{m} + f^{m+1}\right).$$

Crank-Nicolson cont.

When applying Crank-Nicolson to our PDE from earlier, the resulting equation can be expressed in matrix form as such:

$$\left(I + \frac{\Delta t}{2h^{\alpha}} A^{m+1}\right) u^{m+1} = \left(I - \frac{\Delta t}{2h^{\alpha}} A^{m}\right) u^{m} + \frac{\Delta t}{2}\left(f^{m} + f^{m+1}\right).$$

$$A \qquad \vec{x} = \qquad \vec{b}$$

Crank-Nicolson cont.

When applying Crank-Nicolson to our PDE from earlier, the resulting equation can be expressed in matrix form as such:

$$\left(I + \frac{\Delta t}{2h^{\alpha}} A^{m+1}\right) u^{m+1} = \left(I - \frac{\Delta t}{2h^{\alpha}} A^{m}\right) u^{m} + \frac{\Delta t}{2}\left(f^{m} + f^{m+1}\right).$$

$$A \qquad \vec{x} = \qquad \vec{b}$$

What can we use to solve this?

Crank-Nicolson cont.

When applying Crank-Nicolson to our PDE from earlier, the resulting equation can be expressed in matrix form as such:

$$\left(I + \frac{\Delta t}{2h^{\alpha}} A^{m+1}\right) u^{m+1} = \left(I - \frac{\Delta t}{2h^{\alpha}} A^{m}\right) u^{m} + \frac{\Delta t}{2}\left(f^{m} + f^{m+1}\right).$$

$$A \qquad \vec{x} = \qquad \vec{b}$$

What can we use to solve this? Gaussian Elimination & Matrix Inversion.

Crank-Nicolson cont.

When applying Crank-Nicolson to our PDE from earlier, the resulting equation can be expressed in matrix form as such:

$$\left(I + \frac{\Delta t}{2h^{\alpha}} A^{m+1}\right) u^{m+1} = \left(I - \frac{\Delta t}{2h^{\alpha}} A^{m}\right) u^{m} + \frac{\Delta t}{2}\left(f^{m} + f^{m+1}\right).$$

$$A \qquad \vec{x} = \qquad \vec{b}$$

What can we use to solve this? Gaussian Elimination & Matrix Inversion.

$$\vec{x} = A^{-1}\vec{b}$$

Gaussian Elimination & Matrix Inversion

In order to get A^{-1} from A, we must use Gaussian Elimination. An overview of this process:

Gaussian Elimination & Matrix Inversion

In order to get A^{-1} from A, we must use Gaussian Elimination. An overview of this process:

$A \rightarrow$ series of row operations $\rightarrow I$ (identity matrix)

simultaneously:

$I \rightarrow$ **same** series of row operations $\rightarrow A^{-1}$ [9]

Gaussian Elimination & Matrix Inversion

In order to get A^{-1} from A, we must use Gaussian Elimination. An overview of this process:

$A \to$ series of row operations $\to I$ (identity matrix)

simultaneously:

$I \to$ **same** series of row operations $\to A^{-1}$ [9]

Row operations $\to$ **Gaussian Elimination**

Gaussian Elimination & Matrix Inversion

In order to get A^{-1} from A, we must use Gaussian Elimination. An overview of this process:

$A \rightarrow$ series of row operations $\rightarrow I$ (identity matrix)

simultaneously:

$I \rightarrow$ **same** series of row operations $\rightarrow A^{-1}$ [9]

Row operations $\rightarrow$ **Gaussian Elimination**

How efficient is this process?

Gaussian Elimination requires $O(N^3)$ time, demonstrated by counting the number of arithmetic operations to get the necessary numbers in each spot in the matrix. [9]

Gaussian Elimination requires $O(N^3)$ time, demonstrated by counting the number of arithmetic operations to get the necessary numbers in each spot in the matrix. [9]

What does this mean?

Gaussian Elimination requires $O(N^3)$ time, demonstrated by counting the number of arithmetic operations to get the necessary numbers in each spot in the matrix. [9]

What does this mean?

Then we must do $A^{-1} * b$. This process is $O(N^2)$.

Total computational cost: $O(N^3)$. Why?

Conclusion: This is **VERY** inefficient. Can we do better?

Room for Improvement

We saw earlier how matrix inversion is expensive.

Room for Improvement

We saw earlier how matrix inversion is expensive.

$O(N^3)$ computation and $O(N^2)$ memory

Room for Improvement

We saw earlier how matrix inversion is expensive.

$O(N^3)$ computation and $O(N^2)$ memory

To replace this costly inversion process done via Gaussian Elimination, we will use an iterative method known as **The Conjugate Gradient Squared Method**.

$\rightarrow$ the goal is to get the resultant vector at each time step with significantly less computation

What Is An Iterative Method?

Initial guess, x_0, $\rightarrow$ evaluate residual $\rightarrow$ make some modification & generate approximation

residual: $b - Ax_i$

What Is An Iterative Method?

Initial guess, x_0, $\rightarrow$ evaluate residual $\rightarrow$ make some modification & generate approximation

residual: $b - Ax_i$

After every approximation, a residual is calculated and compared to the convergence criteria- also known as the "tolerance". The tolerance used in the test cases was 10^{-6}.

What Is An Iterative Method?

Initial guess, x_0, $\rightarrow$ evaluate residual $\rightarrow$ make some modification & generate approximation

residual: $b - Ax_i$

After every approximation, a residual is calculated and compared to the convergence criteria- also known as the "tolerance". The tolerance used in the test cases was 10^{-6}.

Error analysis is crucial in carrying out an iterative method.

What Is An Iterative Method?

Initial guess, x_0, $\rightarrow$ evaluate residual $\rightarrow$ make some modification & generate approximation

residual: $b - Ax_i$

After every approximation, a residual is calculated and compared to the convergence criteria- also known as the "tolerance". The tolerance used in the test cases was 10^{-6}.

Error analysis is crucial in carrying out an iterative method.

$\rightarrow L^2$ and L^∞ norms.

Trust the Process

CGS iterative process [7]:

$$x_i = x_{i-1} + \alpha r_{i-1}$$

x_i: current solution

x_{i-1}: previous solution generated by CGS

r_{i-1}: the residual vector r (line in the direction of steepest descent)

α: the factor that identifies how far down the line to go

The residual in this case is $b - Ax_i$

CG vs. CGS

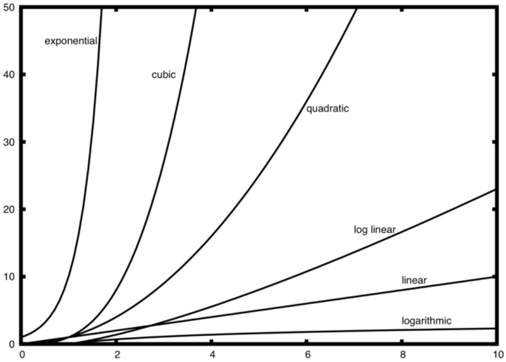
Conjugate Gradient (CG) Method:

- CG requires a symmetric matrix
- simpler algorithm, requires less calculation

Conjugate Gradient Squared (CGS) Method:

- Does not require symmetry
- Requires more scalar computations, including 6 matrix-vector multiplications
- An extended version of CG

Benefit: The CGS method has a computational cost of $O(N^2)$ per time step, as opposed to $O(N^3)$.



Note: did you catch that the matrix A is still being used in the CGS? Still $O(N^2)$ memory.

Less is More, Or Just Better

Claim: A^{m+1} can be stored using only $O(N)$ memory.

Consider the following decomposition:

$$A^{m+1} = \left(d_{+,i}^{m+1}\right) A_L^{m+1} + \left(d_{-,i}^{m+1}\right) A_R^{m+1}$$

Less is More, Or Just Better

Claim: A^{m+1} can be stored using only $O(N)$ memory.

Consider the following decomposition:

$$A^{m+1} = \left(d_{+,i}^{m+1}\right) A_L^{m+1} + \left(d_{-,i}^{m+1}\right) A_R^{m+1}$$

- The diffusion coefficients line the main diagonal of their own matrices

Less is More, Or Just Better

Claim: A^{m+1} can be stored using only $O(N)$ memory.

Consider the following decomposition:

$$A^{m+1} = \left(d_{+,i}^{m+1}\right) A_L^{m+1} + \left(d_{-,i}^{m+1}\right) A_R^{m+1}$$

- The diffusion coefficients line the main diagonal of their own matrices
- A_L^{m+1} and A_R^{m+1} are Toeplitz matrices made up of the Grünwald Weights

Less is More, Or Just Better

Claim: A^{m+1} can be stored using only $O(N)$ memory.

Consider the following decomposition:

$$A^{m+1} = \left(d_{+,i}^{m+1}\right) A_L^{m+1} + \left(d_{-,i}^{m+1}\right) A_R^{m+1}$$

- The diffusion coefficients line the main diagonal of their own matrices
- A_L^{m+1} and A_R^{m+1} are Toeplitz matrices made up of the Grünwald Weights

Definition: Toeplitz matrices are matrices that are constant, left-to-right, along all diagonals.

Definition: Circulant matrices are matrices with the property that each row or column are rotations of another.

Let

$$T_3 = \begin{bmatrix} 1 & 2 & 4 \\ 3 & 1 & 2 \\ 5 & 3 & 1 \end{bmatrix} \quad B_3 = \begin{bmatrix} 0 & 5 & 3 \\ 4 & 0 & 5 \\ 2 & 4 & 0 \end{bmatrix}$$

then the Toeplitz matrix T_n can be embedded into a Circulant matrix C_n with the help of B_n as follows:

$$C_6 = \begin{bmatrix} 1 & 2 & 4 & 0 & 5 & 3 \\ 3 & 1 & 2 & 4 & 0 & 5 \\ 5 & 3 & 1 & 2 & 4 & 0 \\ 0 & 5 & 3 & 1 & 2 & 4 \\ 4 & 0 & 5 & 3 & 1 & 2 \\ 2 & 4 & 0 & 5 & 3 & 1 \end{bmatrix}$$

A circulant matrix C_{2N} can be decomposed as

$$C_{2N} = F_{2N}^{-1} \operatorname{diag}(F_{2N} c) F_{2N}$$

This decomposition also helps reduce the complexity of matrix-vector multiplications:

A circulant matrix C_{2N} can be decomposed as

$$C_{2N} = F_{2N}^{-1}\,\mathrm{diag}(F_{2N}c)F_{2N}$$

This decomposition also helps reduce the complexity of matrix-vector multiplications:

- We want to do $A^{m+1} * u^{m+1}$, which means we will need $C_{2N}^{m+1} * u_{2N}^{m+1}$.

A circulant matrix C_{2N} can be decomposed as

$$\mathsf{C}_{2N} = \mathsf{F}_{2N}^{-1}\ \mathrm{diag}(\mathsf{F}_{2N}\mathsf{c})\mathsf{F}_{2N}$$

This decomposition also helps reduce the complexity of matrix-vector multiplications:

- We want to do $A^{m+1} * u^{m+1}$, which means we will need $C_{2N}^{m+1} * u_{2N}^{m+1}$.
- $F_{2N} * u_{2N}^{m+1}$ can be carried out in $O(N \log N)$ operations via FFT

A circulant matrix C_{2N} can be decomposed as

$$C_{2N} = F_{2N}^{-1}\,\mathrm{diag}(F_{2N}c)F_{2N}$$

This decomposition also helps reduce the complexity of matrix-vector multiplications:

- We want to do $A^{m+1} * u^{m+1}$, which means we will need $C_{2N}^{m+1} * u_{2N}^{m+1}$.
- $F_{2N} * u_{2N}^{m+1}$ can be carried out in $O(N \log N)$ operations via FFT
- $C_{2N}^{m+1} * u_{2N}^{m+1}$ can be evaluated in $O(N \log N)$ operations

A circulant matrix C_{2N} can be decomposed as

$$\mathsf{C}_{2N} = \mathsf{F}_{2N}^{-1}\,\mathrm{diag}(\mathsf{F}_{2N}\mathsf{c})\mathsf{F}_{2N}$$

This decomposition also helps reduce the complexity of matrix-vector multiplications:

- We want to do $A^{m+1} * u^{m+1}$, which means we will need $C_{2N}^{m+1} * u_{2N}^{m+1}$.
- $F_{2N} * u_{2N}^{m+1}$ can be carried out in $O(N \log N)$ operations via FFT
- $C_{2N}^{m+1} * u_{2N}^{m+1}$ can be evaluated in $O(N \log N)$ operations
- $A_{L,N}^{m+1} * u_N^{m+1}$ and $A_{R,N}^{m+1} * u_N^{m+1}$ can be evaluated in $O(N \log N)$ operations

A circulant matrix C_{2N} can be decomposed as

$$C_{2N} = F_{2N}^{-1} \operatorname{diag}(F_{2N} c) F_{2N}$$

This decomposition also helps reduce the complexity of matrix-vector multiplications:

- We want to do $A^{m+1} * u^{m+1}$, which means we will need $C_{2N}^{m+1} * u_{2N}^{m+1}$.
- $F_{2N} * u_{2N}^{m+1}$ can be carried out in $O(N \log N)$ operations via FFT
- $C_{2N}^{m+1} * u_{2N}^{m+1}$ can be evaluated in $O(N \log N)$ operations
- $A_{L,N}^{m+1} * u_{N}^{m+1}$ and $A_{R,N}^{m+1} * u_{N}^{m+1}$ can be evaluated in $O(N \log N)$ operations
- $A^{m+1} * u^{m+1}$ can be evaluated in $O(N \log N)$ operations!

This process brings the computational cost of the CGS from $O(N^2)$ to $O(NlogN)$ per time step.

$$C_{2N} = F_{2N}^{-1}\,\mathrm{diag}(F_{2N}c)F_{2N}$$

Notice how all we need is a diagonal from each circulant matrix? $O(N)$ memory achieved!

Overview:

Decompose A $\rightarrow$ embed Toeplitz parts into Circulant matrices $\rightarrow$ decompose Circulant matrices using FFT $\rightarrow$ evaluate $C_L u$ and $C_R u$

This process brings the computational cost of the CGS from $O(N^2)$ to $O(NlogN)$ per time step.

$$C_{2N} = F_{2N}^{-1} \operatorname{diag}(F_{2N} c) F_{2N}$$

Notice how all we need is a diagonal from each circulant matrix? $O(N)$ memory achieved!

Overview:

Decompose A $\rightarrow$ embed Toeplitz parts into Circulant matrices $\rightarrow$ decompose Circulant matrices using FFT $\rightarrow$ evaluate $C_L u$ and $C_R u$

Reduced memory and computational complexity!

Complexity Analysis- Fast Method

Naive vs. Fast Method:

N=M	Naive Time (s)	Fast Time (s)	Naive ratio	Fast ratio
2^2	3.30×10^{-1}	2.5×10^{-1}		
2^3	4.20×10^{-1}	1.87×10^{-1}	1.27	.75
2^4	1.34×10^{0}	2.34×10^{-1}	3.19	1.25
2^5	7.87×10^{0}	6.55×10^{-1}	5.87	2.80
2^6	5.91×10^{1}	4.82×10^{0}	7.50	7.36
2^7	4.58×10^{2}	2.10×10^{1}	7.76	4.35
2^8	3.65×10^{3}	8.30×10^{1}	7.97	3.96
2^9	2.91×10^{4}	2.53×10^{2}	7.98	3.05
2^{10}	DNF	8.45×10^{2}	NaN	3.34

2.91×10^4 seconds $\sim$ 8 hours $\quad$ 8.45×10^2 seconds $\sim$ 14 minutes

What Did We Accomplish?

Richardson Extrapolation:

- Overall convergence increased from first to second-order in space

Properties of Toeplitz & Circulant Matrices:

- Memory required reduced from $O(N^2)$ to $O(N)$

Iterative Method & FFT Decomposition:

- Time complexity improved from $O(N^3)$ to $O(NlogN)$ per time step.

Future Advances

Although this algorithm handles the one-dimensional case well (variables x and t), most problems in nature are two or three dimensional in space.

$$\text{so } u = u(x, y, t) \text{ or } u = (x, y, z, t)$$

Therefore the CGS Method, Richardson Extrapolation, and fast vector-multiplication technique all need to be applied to these extended differential equations.

Additional Concern:

- Moving boundary problem

Acknowledgments

Special Thanks to:

- Occidental College
- The Library Computers
- Professor Treena Basu

[1] Joseph Klafter and Igor M Sokolov. *Anomalous diffusion spreads its wings*. Physics World. *physicsweb.org*, August, 2005.

[2] Donato Cafagna. *Fractional Calculus: A Mathematical Tool from the Past for Present Engineers*. IEEE Industrial Electronics Magazine, pp. 35-40, 2007.

[3] Treena S. Basu and Hong Wang. *A Fast Second-Order Finite Difference Method for Space-Fractional Diffusion Equations*. International Journal of Numerical Analysis and Modeling, Volume 9, Number 3, pp.658-666, 2011.

[4] Hong Wang and Treena S. Basu. *A Fast Finite Difference Method for Two-Dimensional Space-Fractional Diffusion Equations*. SIAM J. Sci. Comput. Vol. 34, No. 5, pp. A2444-A2458.

[5] H. O. A. Wold and P. Whittle. *A Model Explaining the Pareto Distribution of Wealth*. Econometrica, Volume 25, Number 4, pp. 591-595, Oct 1957.

[6] G. E. Uhlenbeck and L. S. Ornstein. *On the Theory of the Brownian Motion*. Physics Review, Volume 36, Number 5, pp. 823-841, Sep 1930.

[7] Jonathan R. Shewchul. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Carnegie Mellon University, Pittsburgh, PA, 1994.

[8] Gene H.Golub and Charles F.Van Loan. *Matrix Computations*. Johns Hopkins University Press, Third Edition, Oct 1996.

[9] John H.Mathews and Kurtis D.Fink. *Numerical Methods Using Matlab*. Prentice Hall, Upper Saddle River, Third Edition, pp. 147-148, NJ, 1999.

[10] Mark M. Meerschaert, Hans-Peter Scheffler, and Charles Tadjeran. *Finite difference methods for two-dimensional fractional dispersion equation*. Journal of Computational Physics, 211, pp. 249-261, 2006.

[11] Charles Tadjeran and Mark M. Meerschaert. *A second-order accurate numerical method for the two-dimensional fractional diffusion equation*. Journal of Computational Physics, 220, pp. 813-823, 2007.

[12] Phillip J. Davis. *Circulant Matrices*. John Wiley & Sons, New York, 1979.

[13] Robert M. Gray. *Toeplitz and circulant matrices: A review*. Foundations and Trends in Communications and Information Theory, 2, pp. 155-239, 2006.

[14] Mark M. Meerschaert and Charles Tadjeran. *Finite Difference Approximations for Two-Sided Space-Fractional Partial Differential Equations* Applied Numerical Mathematics, Volume 56, Issue 1, pp. 80-90, Jan. 2006.