

# Re: Math library

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-09/msg01371.html>

---

- *From:* "Jon Slaughter" <[Jon\\_Slaughter@xxxxxxxxxxxx](mailto:Jon_Slaughter@xxxxxxxxxxxx)>
  - *Date:* Wed, 12 Sep 2007 13:43:54 -0500
- 

"Ben Voigt [C++ MVP]" <[rbv@xxxxxxxxxxxx](mailto:rbv@xxxxxxxxxxxx)> wrote in message  
[news:%23uLhfaV9HHA.1188@xxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:%23uLhfaV9HHA.1188@xxxxxxxxxxxxxxxxxxxxxxxxxxxx)

"Jon Slaughter" <[Jon\\_Slaughter@xxxxxxxxxxxx](mailto:Jon_Slaughter@xxxxxxxxxxxx)> wrote in message  
[news:pvJFi.14859\\$3x.5207@xxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:pvJFi.14859$3x.5207@xxxxxxxxxxxxxxxxxxxxxxxxxxxx)

I wrote a routine to replace Math's Exp method but it turns out to be almost 2x slower ;/ (well, actually its about 1.5x in release)

Jon, when I suggested a lookup table for your particular problem, I didn't mean for you to make a general-purpose Exp routine implemented with a lookup table. I meant to have a lookup table mapping the 256 possible values that the integer variable appearing in your expression could take, to the value of the whole expression.

Hehe, no. I was writing it because I'm kinda putting together a small vector library so I can do some problems that I've been wanting and are to slow in matlab. I figured that I might as well put together a simple numerical library that uses lookup tables while I'm at it. So I threw together what I thought would have been a better routine than Math.Exp and I was wrong ;/ Actually all I'll probably never use Exp inside a critical loop as its just for initialization but who knows. Its nice to see that Microsoft finally did there homework with .NET.

My real problem is that for a 2D physics problem I have, it requires 4 nested loops just to caculate one period of time. (so technically 5 if I want to evolve the system)

The issue is that to do the simulation I have to take into account boundary values. For example, Suppose I want to take the gradient of a scalar field S. This requires I compute  $(S[i+1, j] - S[i, j])/dx$  or equivalently  $(S[i, j] - S[i-1, j])/dx$  or  $(S[i+1, j] - S[i-1, j])/dx/2$ .

## Re: Math library

So if I'm at the boundary then I have a problem. Doing checks and applying the correct formula would be inefficient.

What I do is when I want to specifically compute the something like this that as to deal with the boundary is break it up into 9 sections and deal with each one separately. But how to codify this in general where there is ultimately only 1 main loop(which might contain sub loops but never duplicating loops).

What I mean suppose I want to compute the gradient then the laplacian.

Both of these depend on S only but have to take into account the boundaries.

Right now I have two functions that essentially loop over S separately.

So if I call

```
Grad(S);  
Laplacian(S);
```

I have 2 loops when in reality I could have "consolidated" them into one. Now this isn't a big deal until you consider that I have to break such a loop into 9 parts to handle the boundaries. (I can wrap the boundaries but I do not want to do that for this problem... I will implement that later as an option... which will be much easier).

The only real option is to codify whatever I'm doing by hand but it gets messy real quick. I do have routines that can compute the gradient and laplacian at only a point instead of computing it on the entire field but it has to check if the point is on the boundary. What this means is that if the user called those function it would be slow because of the checks(a bit easier to use though which is why I included it).

The following is the code I use to compute the laplacian of a scalar field. (I do have another function that actually returns a new scalar field but I want to try to avoid allocating new fields because its slow and could get very messy quick if one starts doing some algebra on the field)

The general scheme of the laplacian is applied to every manipulation of a scalar field. Basically just handle corners, sides, and then inner part. In this case I'm calculating the second derivatives but in reality it could be much more complicated.

The real meat of the function is this which is simple... the problem is the boundaries as they require special techniques.

```
// Inner Matrix
```

```
for (int j = 1; j < SF.Ny - 1; j++)
```

```
for (int i = 1; i < SF.Nx - 1; i++)
```

Re: Math library

Re: Math library

```
{  
x = (SF[i + 1, j] + SF[i - 1, j] - 2 * SF[i, j]) / dx2;  
y = (SF[i, j + 1] + SF[i, j - 1] - 2 * SF[i, j]) / dy2;  
SF2[i, j] = x + y;  
}
```

I was thinking about making a routine where the user could register a callback in each part so they could "hook" into the boundaries and put whatever computation they wanted so they wouldn't have to actually write out all the loops... but all those function calls would be slow unnecessarily I suppose that since each side is mirrored inwards I could optimize it so that there is only "one side" in some sense but I think that might be impossible to actually do because it would depend on figuring how the indicies used in the callback or end up with a lot of checks.

Anyways, chances are I'll have to recode this loop every time I want a tight loop but I feel for my problem that if I can abstract things efficiently that its going to get messy real quick.

Maybe someone has some ideas though?

Thanks,

Jon

```
public unsafe static ScalarField2d Laplacian(ScalarField2d SF, ref  
ScalarField2d SF2)
```

```
{  
  
double x, y, dx2 = SF.dx * SF.dx, dy2 = SF.dy * SF.dy;
```

Re: Math library

Re: Math library

```
SF2.dx = SF.dx; SF2.dy = SF.dy;
```

```
// Evaluate the Laplacian at corners using single sided differences
```

```
x = (-SF[3, 0] + 4 * SF[2, 0] - 5 * SF[1, 0] + 2 * SF[0, 0]) / dx2;
```

```
y = (-SF[0, 3] + 4 * SF[2, 0] - 5 * SF[1, 0] + 2 * SF[0, 0]) / dy2;
```

```
SF2[0, 0] = x + y;
```

```
x = (2 * SF[SF.Nx - 1, 0] - 5 * SF[SF.Nx - 1 - 1, 0] + 4 * SF[SF.Nx - 1 - 2, 0] - SF[SF.Nx - 1 - 3, 0]) / dx2;
```

```
y = (-SF[SF.Nx - 1, 3] + 4 * SF[SF.Nx - 1, 2] - 5 * SF[SF.Nx - 1, 1] + 2 * SF[SF.Nx - 1, 0]) / dy2;
```

```
SF2[SF.Nx - 1, 0] = x + y;
```

```
x = (2 * SF[SF.Nx - 1, SF.Ny - 1] - 5 * SF[SF.Nx - 1 - 1, SF.Ny - 1] + 4 * SF[SF.Nx - 1 - 2, SF.Ny - 1] - SF[SF.Nx - 1 - 3, SF.Ny - 1]) / dx2;
```

```
y = (2 * SF[SF.Nx - 1, SF.Ny - 1] - 5 * SF[SF.Nx - 1, SF.Ny - 1 - 1] + 4 * SF[SF.Nx - 1, SF.Ny - 1 - 2] - SF[SF.Nx - 1, SF.Ny - 1 - 3]) / dy2;
```

```
SF2[SF.Nx - 1, SF.Ny - 1] = x + y;
```

```
x = (2 * SF[0, SF.Ny - 1] - 5 * SF[1, SF.Ny - 1] + 4 * SF[2, SF.Ny - 1] - SF[3, SF.Ny - 1]) / dx2;
```

```
y = (2 * SF[0, SF.Ny - 1] - 5 * SF[0, SF.Ny - 1 - 1] + 4 * SF[0, SF.Ny - 1 - 2] - SF[0, SF.Ny - 1 - 3]) / dy2;
```

```
SF2[0, SF.Ny - 1] = x + y;
```

```
// left and right column
```

```
for (int j = 1; j < SF.Ny - 1; j++)
```

```
{
```

```
x = (-SF[3, j] + 4 * SF[2, j] - 5 * SF[1, j] + 2 * SF[0, j]) / dx2;
```

```
y = (SF[0, j + 1] + SF[0, j - 1] - 2 * SF[0, j]) / dy2;
```

```
SF2[0, j] = x + y;
```

```
x = (2 * SF[SF.Nx - 1, j] - 5 * SF[SF.Nx - 1 - 1, j] + 4 * SF[SF.Nx - 1 - 2, j] - SF[SF.Nx - 1 - 3, j]) / dx2;
```

Re: Math library

## Re: Math library

```
y = (SF[SF.Nx - 1, j + 1] + SF[SF.Nx - 1, j - 1] - 2 * SF[SF.Nx - 1, j]) /  
dy2;
```

```
SF2[SF.Nx - 1, j] = x + y;
```

```
}
```

```
// top and bottom row
```

```
for (int i = 1; i < SF.Nx - 1; i++)
```

```
{
```

```
x = (SF[i + 1, 0] + SF[i - 1, 0] - 2 * SF[i, 0]) / dx2;
```

```
y = (-SF[i, 3] + 4 * SF[i, 2] - 5 * SF[i, 1] + 2 * SF[i, 0]) / dy2;
```

```
SF2[i, 0] = x + y;
```

```
x = (SF[i + 1, SF.Ny - 1] + SF[i - 1, SF.Ny - 1] - 2 * SF[i, SF.Ny - 1]) /  
dx2;
```

```
y = (-SF[i, SF.Ny - 1 - 3] + 4 * SF[i, SF.Ny - 1 - 2] - 5 * SF[i, SF.Ny -  
1 - 1] + 2 * SF[i, SF.Ny - 1]) / dy2;
```

```
SF2[i, SF.Ny - 1] = x + y;
```

```
}
```

```
// Inner Matrix
```

```
for (int j = 1; j < SF.Ny - 1; j++)
```

```
for (int i = 1; i < SF.Nx - 1; i++)
```

```
{
```

```
x = (SF[i + 1, j] + SF[i - 1, j] - 2 * SF[i, j]) / dx2;
```

```
y = (SF[i, j + 1] + SF[i, j - 1] - 2 * SF[i, j]) / dy2;
```

```
SF2[i, j] = x + y;
```

```
}
```

```
return SF2;
```

```
} // Laplacian
```

Re: Math library