Geometry-driven diffusion

Initialization

The 1-D Gaussian kernel:

```
gauss[x_, σ_ /; σ > 0] := 1/(σ Sqrt[2 π]) E^(-x^2/(2 σ^2));
```

$$\text{gauss}[x\_, \sigma\_ /; \sigma > 0] := \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}};$$

The universal 2D multi-scale Gaussian derivative function for discrete data **gD[]**:

```
gD[im_List, nx_, ny_, σ_] :=
 Module[{x, y, kx, ky, mid, tmp},
   kx = N[Table[Evaluate[D[gauss[x, σ], {x, nx}]],
    {x, -6 σ, 6 σ}]];
  ky = If[nx == ny, kx,
     N[Table[Evaluate[D[gauss[y, σ], {y, ny}]],
    {y, -6 σ, 6 σ}]]]; mid = Ceiling[Length[#1] / 2] & ;
  tmp = Transpose[
     ListConvolve[{kx}, im, {{1, mid[kx]}, {1, mid[kx]}}]];
  Transpose[ListConvolve[{ky}, tmp,
     {{1, mid[ky]}, {1, mid[ky]}}]]];
```

Options for ArrayPlot to plot properly:

```
SetOptions[ArrayPlot, ColorFunction → GrayLevel, Frame → False];
SetOptions[ListPlot, Joined -> True];
url = "http://bmia.bmt.tue.nl/Education/Courses/FEV/book/images/";
shortnotation[expr_] := Module[{nx, ny, nz, L, x, y, z},
   Simplify[expr /. Derivative[nx_, ny_][L_][x_, y_] ⧴
       Subscript[ToString[L], StringJoin[
         Table[ToString[x], {nx}], Table[ToString[y], {ny}]]]
      /. Derivative[nx_, ny_, nz_][L_][x_, y_, z_] ⧴
       Subscript[ToString[L], StringJoin[Table[ToString[x], {nx}],
         Table[ToString[y], {ny}], Table[ToString[z], {nz}]]]]]];
```

The Perona & Malik Equation

Perona and Malik [Perona and Malik 1990] proposed to make c a function of the gradient magnitude in order to reduce the diffusion at the location of edges:

$$\frac{\partial L}{\partial s} = \vec{\nabla}.c\left(\left|\vec{\nabla}L\right|\right)\vec{\nabla}L$$

with c:

$$c = e^{-\frac{|\vec{\nabla} L|^2}{k^2}}.$$

```
im = ImageData[ColorConvert[Import[
    "http://bmia.bmt.tue.nl/Education/Courses/FEV/book/images/mr256.
      gif"], "Grayscale"], "Byte"];
σ = 2;
GraphicsRow[
  (ArrayPlot[ⅇ^(-(gD[im,1,0,σ]^2+gD[im,0,1,σ]^2)/#1^2), PlotLabel → "k = " <> ToString[#1]] &) /@
   {5, 10, 20, ∞}, ImageSize → 800]
```

The conductivity coefficient c in the P&M equation as a function of the parameter k.

The Perona and Malik (P&M) equation becomes

$$\frac{\partial L}{\partial s} = \vec{\nabla}.\left(e^{-\frac{|\vec{\nabla} L|^2}{k^2}} \vec{\nabla} L\right)$$

Expanding the differential operators for the right hand side, we get in 1D:

```
∂x (Exp[-(∂x L[x])^2/k^2] ∂x L[x]) // Simplify
```

and in 2D:

```
k =.;
PM = ∂x (E^(-((∂x L[x,y])^2+(∂y L[x,y])^2)/k^2) ∂x L[x, y]) +
    ∂y (E^(-((∂x L[x,y])^2+(∂y L[x,y])^2)/k^2) ∂y L[x, y]) // FullSimplify;
PM // shortnotation
```

The most straightforward numerical approximation of $\frac{\partial L}{\partial s} = \nabla.c\nabla L$ is the *forward-Euler* approximation $\delta L = \delta s\,(\nabla.c\nabla L)$.

For the limit $k \to \infty$, we get the linear diffusion equation again:

```
Limit[PM, k -> ∞] // shortnotation
```

Implementation:

```
Clear[im, σ, k];

c = E^-(((∂x L[x,y])^2 + (∂y L[x,y])^2)/k^2);

pm[im_, σ_, k_] = ∂x (c ∂x L[x, y]) + ∂y (c ∂y L[x, y]) /.
   Derivative[n_, m_][L][x_, y_] → gD[im, n, m, σ] // Simplify
```

We calculate the variable conductance diffusion first on a simple small (64x64) noisy test image of a black disk (minimum: 0, maximum: 255):

```
imdisk = Table[If[(x - 32)^2 + (y - 32)^2 < 300, 0, 255], {y, 64}, {x, 64}];
noise = Table[100 RandomReal[], {64}, {64}];
imdn = imdisk + noise;
ArrayPlot[imdn, ImageSize → 500]
```

A rule of thumb for *k* is 80% of the maximal edge strength:

```
Histogram[Flatten[grad = √(gD[imdn, 1, 0, 1]^2 + gD[imdn, 0, 1, 1]^2)],
 ImageSize → 500, PlotRange -> All]
```

Forward-Euler approximation scheme:

```
peronamalik[im_, δs_, σ_, k_, niter_] := Module[{}, evolved = im;
Do[evolved += δs (pm[evolved, σ, k]), {niter}];
   evolved];
```

where **im** is the input image, **δs** is the time step, **σ** is the scale of the differential operator, **k** is the conductivity control parameter and **niter** is the number of iterations. Here is an example of its performance:

```
line = {Red, Line[{{0, 32}, {64, 32}}]};
GraphicsGrid[{(ArrayPlot[#1, Epilog → line] &) /@
   {imdn, imp = peronamalik[imdn, 0.1, 0.7, 25, 40]},
  ListPlot /@ {imdn[[32]], imp[[32]]}}, ImageSize → 600]
```

We study the signal-to-noise ratio (SNR) over time:

```
snr[im_] := Module[{}, m1 = Take[im, {24, 40}, {24, 40}] // Flatten;
m2 = Take[im, {3, 19}, {3, 19}] // Flatten;
    (Mean[m2] - Mean[m1]) / (Variance[m1] + Variance[m2])];
```

```
ArrayPlot[imdn, Epilog →
  {Hue[1], Thick, Line[{{3, 3}, {3, 19}, {19, 19}, {19, 3}, {3, 3}}],
   Line[{{24, 24}, {24, 40}, {40, 40}, {40, 24}, {24, 24}}]},
 ImageSize → 150]
```

Clearly, the signal-to-noise ratio increases substantially during the evolution until $t = \text{niter} \times \ \delta\text{s} = 1$:

```
evolved = imdn;
out = {};
σ = 0.9;
δs = 0.1;
k = 100;
niter = 20;
Do[evolved += δs pm[evolved, σ, k];
  out = Append[out, snr[evolved]], {niter}];
ListPlot[out, Joined → True, AxesLabel →
  {"evolution\ntime\n(in iterations)", "SNR"}, ImageSize → 250]
```

The signal-to-noise ratio (SNR) increases substantially with increasing evolution time.

But this cannot continue, of course, for physical reasons. When we continue the evolution until $t = 100$ (in units of iterations), we see that the gain is lost again:

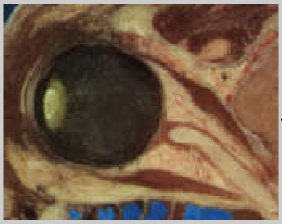
```
evolved = imdn;
out = {};
σ = 0.9;
δs = 0.1;
k = 100;
niter = 100;
Do[evolved += δs pm[evolved, σ, k];
  out = Append[out, snr[evolved]], {niter}];
ListPlot[out, Joined → True, AxesLabel →
  {"evolution\ntime\n(in iterations)", "SNR"}, ImageSize → 250]
```

There is a maximum in the signal-to-noise ratio (SNR) for variable conductance diffusion with increasing evolution time.

```
im = ImageData[
   ColorConvert[Import[url <> "Utrecht256.gif"], "Grayscale"], "Byte"];
```

```
δs = 0.1;
σ = 1;
k = 25;
evolved = im;
Do[evolved += δs pm[evolved, σ, k], {20}];
GraphicsRow[(ArrayPlot[#1] &) /@ {im, evolved}, ImageSize → 800]
```

In *Mathematica* the function **PeronaMalikFilter[ ]** does the anisotropic filtering:

```
PeronaMalikFilter[
```

```
, 100]
```