The following is roughly copied from

https://github.com/usnistgov/fipy/blob/70b72b7abb267c85ada47886b8b3573e1819fffc/documenta
70b72b7abb267c85ada47886b8b3573e1819fffc

The Robin condition

$$\hat{n} \cdot (\vec{a}\phi + b\nabla\phi) = g \qquad \text{on } f = f_0$$

can often be substituted for the flux in an equation

$$\frac{\partial \phi}{\partial t} = \nabla \cdot (\vec{a}\phi) + \nabla \cdot (b\nabla\phi)$$
$$\int_V \frac{\partial \phi}{\partial t}\, dV = \int_S \hat{n} \cdot (\vec{a}\phi + b\nabla\phi)\; dS$$
$$\int_V \frac{\partial \phi}{\partial t}\, dV = \int_{S \neq f_0} \hat{n} \cdot (\vec{a}\phi + b\nabla\phi)\; dS + \int_{f_0} g\, dS$$

```
>>> convectionCoeff = FaceVariable(mesh=mesh, value=[a])
>>> convectionCoeff.setValue(0., where=mask)
>>> diffusionCoeff = FaceVariable(mesh=mesh, value=b)
>>> diffusionCoeff.setValue(0., where=mask)
>>> eqn = (TransientTerm() == PowerLawConvectionTerm(coeff=convectionCoeff)
...         + DiffusionTerm(coeff=diffusionCoeff)
...         + (g * mask * mesh.faceNormals).divergence)
```

When the Robin condition does not exactly map onto the boundary flux, we can attempt to apply it term by term. The Robin condition relates the gradient at a boundary face to the value on that face, however :term:‘FiPy‘ naturally calculates variable values at cell centers and gradients at intervening faces. Using a first order upwind approximation, the boundary value of the variable can be put in terms of the neighboring cell value and the normal gradient at the boundary:

where

$$\vec{d}_{fP}$$

is the distance vector from the face center to the adjoining cell center. The approximation

$$\left(\vec{d}_{fP} \cdot \nabla\phi\right)_{f_0} \approx (\hat{n} \cdot \nabla\phi)_{f_0} \left(\vec{d}_{fP} \cdot \hat{n}\right)_{f_0}$$

is most valid when the mesh is orthogonal.

Substituting this expression into the Robin condition:

we obtain an expression for the gradient at the boundary face in terms of its neighboring cell. We can, in turn, substitute this back into upwind equation to obtain the value on the bounary face in terms of the neighboring cell. Substituting robin facegrad into the discretization of the class fipy DiffusionTerm:

$$\int_V \nabla \cdot (\Gamma \nabla \phi)\, dV \approx \sum_f \Gamma_f \left(\hat{n} \cdot \nabla \phi\right)_f A_f$$
$$= \sum_{f \neq f_0} \Gamma_f \left(\hat{n} \cdot \nabla \phi\right)_f A_f + \Gamma_{f_0} \left(\hat{n} \cdot \nabla \phi\right)_{f_0} A_{f_0}$$
$$\approx \sum_{f \neq f_0} \Gamma_f \left(\hat{n} \cdot \nabla \phi\right)_f A_f + \Gamma_{f_0} \frac{g - \hat{n} \cdot \vec{a} \phi_P}{-\left(\vec{d}_{fP} \cdot \vec{a}\right)_{f_0} + b} A_{f_0}$$

An equation of the form

```
>>> eqn = TransientTerm() == DiffusionTerm(coeff=Gamma0)
```

can be constrained to have a Robin condition at a face identifed by mask by making the following modifications

```
>>> Gamma = FaceVariable(mesh=mesh, value=Gamma0)
>>> Gamma.setValue(0., where=mask)
>>> dPf = FaceVariable(mesh=mesh,
...                    value=mesh._faceToCellDistanceRatio * mesh.cellDistanceVectors)
>>> Af = FaceVariable(mesh=mesh, value=mesh._faceAreas)
>>> RobinCoeff = (mask * Gamma0 * Af * mesh.faceNormals / (dPf.dot(a) + b)).divergence
>>> eqn = (TransientTerm() == DiffusionTerm(coeff=Gamma) + RobinCoeff * g
...        - ImplicitSourceTerm(coeff=RobinCoeff * mesh.faceNormals.dot(a)))
```

Similarly, for a fipy ConvectionTerm, we can substitute upwind2:

$$\int_V \nabla \cdot (\vec{u} \phi)\, dV \approx \sum_f \left(\hat{n} \cdot \vec{u}\right)_f \phi_f A_f$$
$$= \sum_{f \neq f_0} \left(\hat{n} \cdot \vec{u}\right)_f \phi_f A_f + \left(\hat{n} \cdot \vec{u}\right)_{f_0} \frac{-g\left(\hat{n} \cdot \vec{d}_{fP}\right)_{f_0} + b\phi_P}{-\left(\vec{d}_{fP} \cdot \vec{a}\right)_{f_0} + b} A_{f_0}$$

Note
An expression like the heat flux convection boundary condition

$$-k \nabla T \cdot \hat{n} = h(T - T_\infty)$$

can be put in the form of the Robin condition used above by letting

$$\vec{a} \equiv h\hat{n}$$

$$b \equiv k$$

$$g \equiv hT_\infty$$