License Agreement

The Aminpro™ Add-in for Microsoft™ Excel™ ("the Add-In") was developed for internal use by Aminpro and affiliated companies and is offered free for public, non-commercial use.

By installing and using the Add-in, you agree not to hold us responsible for errors, omissions, or usage. You also agree not to redistribute it to third parties, or otherwise make the Add-In publicly available. You agree not to resell the Add-In. We encourage the not-for-profit distribution of the Add-in, but request, for security purposes, that it is done by linking to the web page given below. This enables us to ensure that the most up-to-date version is being used, and helps reduce virus threats.

Installation Instructions

- Create a folder on C:/ root directory called Aminpro. The path should be: C:/Aminpro/
- Copy and paste the Aminpro Add-in.xla file into the new folder.

For Excel 2003

- Open Excel 2003 and Select "Tools > Add-Ins > Browse" and browse the Aminpro Add-In.xla file
- Click Ok.

For Excel 2007 or Excel 2010

- Click the Office Button (the big round decoration in the top left of the Excel screen). This opens the Office Menu. Click the Excel Options button at the bottom of this menu. Click the Add-Ins item in the list along the left edge of the dialog to see the Add-Ins panel. Make sure that the Manage dropdown at the bottom shows Excel Add-Ins, and then press the Go button. This will bring up the familiar Add-Ins dialog. Browse to the Aminpro Add-In.xls file and click OK.

For later versions of Excel:

- On the ribbon click "File > Options > Add-ins"
- At the bottom of the dialog box ensure select Manage Excel add-ins and press "Go…" This will bring up the familiar Add-Ins dialog. Browse to the Aminpro Add-In.xls file and click OK.

Updates

The Aminpro Add-in is updated periodically. The most current version can be downloaded from www.ameltda.com/add-in.php.

Have a suggestion? Just send us a spreadsheet with an example of the calculation and we will consider it for the next version of the add-in. You will receive an acknowledgement in the VBA code.

Function List

Flotation Functions

Notes

1. All recovery input values should be in decimal format (i.e. fractions and not percent). Some functions contain minor data validation checks that are not included in the descriptions below.

=Recovery(head, con, tail)

Calculates metal or other assay value recovery using the three-product formula

Recovery = con / head * (head - tail) / (con - tail)

=CMFlotRec(K, t, Rmax, Rf)

Returns metal recovery for a continuous mixer, excluding entrainment

CMFlotRec = Rc * Rf / (Rc * Rf + 1 - Rc)

Where:

Rc = (K * t) / (1 + K * t) * Rmax

=PFFlotRec(K, t, Rmax, Rf)

Returns metal recovery for a plug flow reactor (batch test), excluding entrainment

PFFlotRec = Rmax (1 – exp(-k * t * Rf))

=CMTotRec(K, t, Rmax, Rf, Re)

Returns metal recovery for a continuous mixer, including entrainment

CMTotRec = (Rflot + ((1 - Rflot) * Rent))

Where:

Rc = (K * t) / (1 + K * t) * Rmax

Rflot = Rc * Rf / (Rc * Rf + 1 - Rc)

=BTRecovery(K, t, Rmax, Rf, Rw, DE)

Returns metal recovery for a batch test, including entrainment

Rflot = Rmax * (1 - Exp(-K * t * Rf))

BTRecovery = Rflot + (1 – Rflot) * Rw * DE

=MassRec(head, con, tail)

Returns mass recovery from metal assays

MassRec = (head - tail) / (con - tail)

=DegEnt(DE0, DE20, DE50, size)

Returns degree of entrainment for a given size from the DE0, DE50, and DE20,

Using the modified Swebrec function

DegEnt = fx / (1 + fx)

Where

fx = Mx ^ (Log(0.25) / Log(M80))

Mx = Log(DE0 / size) / Log(DE0 / DE50)

M80 = Log(DE0 / DE20) / Log(DE0 / DE50)

= Rep(Up, d, v)

Returns the reynolds number for a particle in a fluid

Rep = Up * d / v

Where Up is the settling or rise velocity,

D is the particle diameter, and

V is the fluid viscosity

Comminution Functions

=SAGMillPower(Diameter, EGL, BallCharge, TotalCharge, Percent_of_Critical, Ore_SG)

Returns the SAG mill power draw at the shell, in kW.

c1 = 3.51038

c2 = 2.55086

c3 = 0.57906

c4 = 0.79952

SAGMillPower = c1 * Diameter ^ c2 * EGL * (0.8 * Ore_SG + 0.6 * (BallCharge / 100 * (7.9 - Ore_SG) / (TotalCharge / 100)) + 0.2) * (TotalCharge / 100) ^ c3 * (Percent_of_Critical) ^ c4

= SMCWa(Mia, F80, K1)

'Returns the SAG mill specific energy required to achieve a 750 micron transfer size from the drop weight index using the SMC energy equation

x1 = F80

x2 = 750

e1 = -(0.295 + x1 / 1000000)

e2 = -(0.295 + x2 / 1000000)

SMCWa = Mia * 4 * K1 * (x2 ^ e2 - x1 ^ e1)

= SMCWb(Mib, P80)

'Returns the ball mill specific energy required to achieve a final P80 starting from a grind of 750 microns, using the Mib value

x1 = 750

x2 = P80

e1 = -(0.295 + x1 / 1000000)

e2 = -(0.295 + x2 / 1000000)

SMCWb = Mib * 4 * (x2 ^ e2 - x1 ^ e1)

= SMCWc(Mic As Long, open_circuit As Boolean, HPGR As Boolean, F80 As Long, P80 As Long)

Returns the crusher specific energy required to achieve a final P80 starting from a grind of 750 microns

S = Ks * (F80 * P80) ^ -0.2

x1 = F80

x2 = P80

e1 = -(0.295 + x1 / 1000000)

e2 = -(0.295 + x2 / 1000000)

SMCWc = S * K2 * Mic * 4 * (x2 ^ e2 - x1 ^ e1)

Where

K2 = 1.19 for open circuit

K2 = 1.00 for closed circuit

Ks = 55 for crushers

Ks = 35 for HPGRs

=BallMillPower(Diameter, EGL, BallCharge, Percent_of_Critical)

Returns dry metric tonnes/hr using the Bond equation

ratio = EGL / Diameter

c1 = 0.23584

c2 = 3.672

c3 = 1.38614

c4 = 0.94434

c5 = 1.17798

BallMillPower = c1 * Diameter ^ c2 * ratio ^ c3 * BallCharge ^ c4 * Percent_of_Critical ^ c5

=BondTPH(Wi, P80, F80, CFnet, power)

Returns dry metric tonnes/hr using the Bond equation

BondTPH = power / (CFnet * 10 * Wi * (1 / Sqr(P80) - (1 / Sqr(F80))))

=BondP80(Wi, F80, CFnet, Power, TPH)

Returns the P80 of a ball mill circuit using the Bond equation

BondP80 = 1 / ((F80 ^ -0.5) + (Power / (10 * TPH * Wi * CFnet))) ^ 2

=BondSE(Wi, P80, F80, CFnet)

Returns kWh/tonne using the Bond equation

BondSE = 10 * Wi * CFnet * (1 / sqrt(F80) - 1 / sqrt(P80))

= Vc(diameter, u)

Returns the critical velocity of a tumbling mill, in RPM. u specificies units (metric is default)

Vc = 76.64 / Sqr(diameter)

= MillRPM(diameter, Vpc, u)

Returns the RPM of a tumbling mill, given the diameter and percent of critical speed (Vpc). U specifies units (metric is default)

Vcr = 76.64 / Sqr(diameter) * Vpc

= BondWi(sreensize, grams_per_rev, P80, F80)

Returns the Bond work index, in metric tonnes, from the Bond test data

Wi = 44.5 / (ScreenSize ^ 0.23 * grams_per_rev ^ 0.82 * (10 / (P80 ^ 0.5) - 10 / (F80 ^ 0.5)))

BondWi = 2204.6/2000 * Wi

= SIE(alpha_0, alpha_1, alpha_2, crit_size, size)

Returns the SIE (specific energy breakage function) from the alpha1, alpha2, alpha3, and critical size parameters.

SIE = alpha_0 * ((size / crit_size) ^ alpha_1) / (1 - ((size / crit_size) ^ alpha_2))

=SAGDischarge(SIE, Cum_Perc_Ret, Spec_Energy, PowerNum)

Returns the cumulative percent retained in a given size class in the SAG mill product, from the SIE, cumulative percent retained int eh feed, specific energy, and SAG mill power number.

If PowerNum < 1 Then PowerNum = 1

Min_dc = 0

Max_dc = 1

SAGDischarge = 1 - Cum_Perc_Ret * (1 + SIE * Spec_Energy / PowerNum) ^ (-PowerNum)

= CrusherDischarge(SIE, Cum_Perc_Ret, Spec_Energy)

Returns the cumulative percent retained in a given size class of a cone crusher discharge

CrusherDischarge = 1 - Cum_Perc_Ret * Exp(-SIE * Spec_Energy)

= SGISpEnergy(SGI, Fsag, T80)

Returns the SGI specific energy from the SGI (minutes), Fsag, and T80 (microns)

SGISpEnergy = 5.9 * (SGI / T80 ^ 0.5) ^ 0.55 * Fsag

= SGIT80(SGI, Fsag, E_esp)

Returns the T80 from the specific energy (kWh/mt), SGI, and Fsag

SGIT80 = (SGI / (E_esp / (Fsag * 5.89)) ^ (1 / 0.55)) ^ 2

= t10(a, b, Ecs)

Returns the t10 from the a, b and Ecs parameters

t10 = a * (1 - Exp(-b * Ecs))

Gravity SeparationFunctions

=GravityPartition(size, Density, Pivot_Bypass, Pivot_Density, Viscosity, Flow)

Returns shaking table partion number for a particle of Size and Density using the Rao 2005 Weibull model

GravityPartition = 100 * (1 - Exp(-(Log(1 / (1 - Pivot_Bypass))) * (Density / Pivot_Density) ^ (Viscosity * size ^ Flow)))

Geology, Mineralogy Chemistry Functions

Notes

1. All formula inputs should be in the form of text strings (i.e. in quotations if direct input).

=ChemForm(“mineral” as string)

Returns the chemical formula for the user-specified mineral name

=SpecGrav(mineral_formula)

Returns the specific gravity for the user-specfied mineral formula (as string)

=MolWeight(mineral_formula)

Returns the molecular weight of the user-specfied mineral formula (as string)

=MinFrac(element, mineral_formula)

Returns the % of an element comprising the user-specfied mineral formula (as string)

=AtomicWeight(“A”)

Returns the atomic weight of an element (element in quotations)

=DrillCoreDiam(Size_Code)

Returns the diamond drill core diameter from common drill sizes

Whiten = (Exp(alpha * size / d50) - 1) / (Exp(alpha * size / d50) + Exp(alpha) - 2) * 100

Pumping Functions

= LtoS(PSm)

Returns liquids to solids ratio from the percent solids of a slurry

LtoS = (100 - PSm) / PSm

LtoS = (1 - PSm) / PSm

=PSm(LtoS)

Returns the percent solids from the L:S ratio

PSm = 100 / (1 + LtoS)

=PSv(PSm, SGs)

Returns the percent solids by volume from the SG of the solids and the percent solids by mass

PSv = PSm / SGs * (1 / (100 - PSm + PSm / SGs))

=PulpDensity(PSm, SGs)

Returns the pulp density from solids SG and the percent solids by mass for water slurry

PulpDensity = 1 / (PSm / SGs + (1 - PSm))

=PulpPerSol(PulpDensity, SolidsSG)

Returns the pulp percent solids from the pulp density and solids specific gravity

PulpPerSol = (1 / PulpDensity - 1) / (1 / SolidsSG - 1)

= PulpPerSol2(SGPulp, SGSolid, SGLiquid)

Returns the pulp percent solids from the density solids specific gravity, and liquids specific gravity

PulpPerSol2 = (SGSolid * SGLiquid - SGPulp * SGSolid) / (SGPulp * SGLiquid - SGPulp * SGSolid)

=FrictionLoss(HazenWilliamsC, Q, ID, EqLength)

Returns friction losses in a pipe using ^ Hazen-Williams equation (units are metric)

FrictionLoss = EqLength * 0.002083 * (100 / HazenWilliamsC) ^ 1.85 * (Q * 1000 / 3.785 / 60) ^ 1.85 / (ID * 100 / 2.54) ^ 4.8655

=CriticalVelocity(PSv, SGs, D50, ID)

Returns critical settling velocity for a slurry

CriticalVelocity = 1.25 * (-0.0044 + PSv * 0.0034 + Log(D50) * 0.182975) * (2 * 9.81 * ID * (SGs - 1)) ^ 0.25

=WaterVisc(Temp)

Returns water viscosity as a function of the temperature (in degrees Kelvin)

A = 0.00002414

b = 247.8

C = 140

WaterVisc = A * 10 ^ (b / (Temp - C)) * 1000

=PulpViscosity(PSm, SGs, Temp)

Returns slurry viscosity from the percent solids and solids specific gravity for a given temperature (Kelvin)

Vw = WaterVisc(Temp)

PSVol = PSv(PSm, SGs)

PulpViscosity = Vw / (1 - PSVol ^ 0.333)

=PulpViscosityCR(PSm, SGs, Temp)

Returns slurry viscosity from the percent solids and solids specific gravity for a given temperature (Kelvin)

Vw = WaterVisc(Temp)

PSVol = PSv(PSm, SGs)

PulpViscosity = Vw * exp(2.5 * PSVol / (1-(0.609 * PSVol)))

Size Distribution Functions

= GGSK50(D50, m)

Returns the Gates-Gaudin-Schuhman constant K from the D50 and constant m

GGSK = D50 / (0.5) ^ (1 / m)

= GGSK80(D80, m)

Returns the Gates-Gaudin-Schuhman constant K from the D80 and constant m

GGSK = D80 / (0.8) ^ (1 / m)

= GGSP80(K, m)

Returns the Gates-Gaudin-Schuhman P80 from the K and m constants

GGSP80 = 0.8 ^ (1 / m) * K

=GGSX(K, m, P)

Uses the Gates-Gaudin-Schuhman distribution to calculate the size for a given percent passing value

GGSX = (P ^ (1 / m)) * K

= GGSP(K, m, X)

Uses the Gates-Gaudin-Schuhman distribution to return the percent passing size X

GGSPx = (X / K) ^ m

= RosRamPX(b, m, percent)

Returns Px size (size at which x percent is finer) using Rosin-Rammler equation

RosRamPX = (Log(1 - percent / 100) / -b) ^ (1 / m)

= RosRamB(b, m, size)

Returns cumulative percent passing size x from Rosin-Rammler b and m constants

RosRamB = (1 - Exp(-b * (size ^ m))) * 100

= RosRamP80(P80, m, size)

Returns cumulative percent passing size x from P80 and Rosin Rammler m constant

RosRamP80 = (1 - Exp(Log(0.2) / (P80 ^ m) * size ^ m)) * 100

= RosRamP80X(P80, m, percent)

Returns size at which the user-specified percent passing occurs, using P80 and m values

RosRamP80X = (P80 ^ m * Log(1 - percent / 100) / Log(0.2)) ^ (1 / m)

=Swebrec(xmax, x50, b, size)

Returns cumulative percent passing size x from Swebrec function

Swebrec = 100 / (1 + (Log(xmax / size) / Log(xmax / x50)) ^ b)

=SwebrecX(xmax, x50, b, percent)

Returns size X for the given percent passing using the Swebrec function

Swebrecx = xmax / Exp(Log(xmax / X50) * (1 / percent - 1) ^ (1 / b))

=SwebrecP80(x100, x80, x50, size)

Uses the Swebrec function to returns the cumulative percent passing “size” given the X100, X80, and X50 of a size distribution

Mx = Log(X100 / size) / Log(X100 / X50)

M80 = Log(X100 / X80) / Log(X100 / X50)

SwebrecP80 = 1 / (1 + Mx ^ (Log(0.25) / Log(M80))) * 100

=SwebrecP80X(x100, x80, x50)

Uses the Swebrec function to return the size at which the user-specified percent passing occurs, using the X100, X80, and X50 of a size distribution

M80 = Log(X100 / X80) / Log(X100 / X50)

b = Log(0.25) / Log(M80)

SwebrecP80X = X100 / Exp(Log(X100 / X50) * (1 / percent - 1) ^ (1 / b))

=SwebrecB(x100, 80, bx50,)

Returns the Swebrec b parameter from the X100, X50, and X80 parametesr.

M80 = Log(X100 / X80) / Log(X100 / X50)

SwebrecB = Log(0.25) / Log(M80)

=TylerSize(mesh_number)

Returns the Tyler sieve size in microns for a given mesh number

=ASTMSize(sieve_number)

Returns the ASTM E11:87 sieve size in microns for the given sieve number

Classification Functions

=Whiten(D50, alpha, size)

Returns cumulative percent passing size x from Whiten function

Whiten = (Exp(alpha * size / d50) - 1) / (Exp(alpha * size / d50) + Exp(alpha) - 2) * 100

=Plitt(d50, m, size)

Returns corrected recovery to U/F using Plitt function

Plitt = (1 - Exp(-Log(2) * (size / d50) ^ m)) * 100

=WhitenB(d50, alpha, beta, size)

Returns the cumulative percent passing (or recovery) using the modified Whiten function

WhitenB = (Exp(alpha * size / d50) - beta) / (Exp(alpha * size / d50) + Exp(alpha) - (2 * beta)) * 100

=WhitenBX(d50, alpha, beta, percent)

Returns the size at which a user-specified percent passing or recovery ocurrs, modified Whiten.

WhitenBX = (d50 / alpha) * Log((percent * -Exp(alpha) + (percent * 2 * beta) - beta) / (percent - 1))

=Krebs_Cap(Cyc_Model, VortexFinder_Diam, Pressure)

Returns feed capacity of common Krebs cyclones given a vortex finder and inlet pressure. The cyclone model must be selected from the following list, and the vortex finder diameter must be within the minimum and maximum given.

Description	VF_min	VD_median	VF_max
gMAX4	0.75	1.13	1.50
gMAX6-10	1.25	1.88	2.50
gMAX10	2.00	3.00	4.50
gMAX10-20	2.00	3.25	4.50
D15B-11 In	4.00	4.88	6.00
D15B-13 In	4.00	4.88	6.00
D15LB-gMAX	4.00	5.25	6.75
gMAX15	3.50	4.88	6.75
gMAX15-20	4.00	5.25	6.75
gMAX20	6.00	7.50	9.00

gMAX20-20	6.00	7.50	9.00
gMAX20-H	6.00	7.50	9.00
gMAX26	6.50	9.00	12.00
gMAX26-20	6.50	9.00	12.00
gMAX26-30	6.50	9.00	12.00
gMAX26-H	6.50	9.00	12.00
gMAX33	10.00	13.00	16.00
gMAX33-20	10.00	13.00	16.00
gMAX33-30	10.00	13.00	16.00
gMAX33-H	10.00	13.00	16.00

Dewatering and Tailings

= VsLee(rho_l, rho_s, mu_l, perc_sol, d)

Uses Lee's method of determining the terminal hindered settling velocity of a particle in a fluid. Inputs are: density of the liquid, density of the particle, viscosity of the liquid, percent solids of the slurry, and diameter of the particle.

```
f1 = (1 + 0.75 * percsol ^ 0.333) * (1 - percsol) * (1 - 1.47 * percsol + 2.67 * percsol ^ 2) ^ 2 / _
    ((1 - 1.45 * percsol) ^ 1.83 * (1 + 2.25 * percsol ^ 3.7))

f2 = (1 + 2.25 * percsol ^ 3.7) * (1 - 1.45 * percsol) ^ 1.83 / _
    ((1 + 0.75 * percsol ^ 0.333) * (1 - percsol) * (1 - 1.47 * percsol + 2.67 * percsol ^ 2))

rho_p = PulpDensity(percsol, rho_s)

VsLee = (20.52 * mu_l) / (d * rho_p) * f1 * ((1 + 0.0921 * ((d ^ 3 * Abs(rho_s - rho_p) * rho_p * 9.81) /
(0.75 * mu_l ^ 2)) ^ 0.5 * f2) ^ 0.5 - 1) ^ 2
```

= WilhelmNaideUA(a, b, Cs)

Determines the required thickener specific area (UA) using the Wilhelm & Naide equation

```
WilhelmNaideUA = (((b - 1) / b) ^ (b - 1)) / (a * b) * (Cs ^ (b - 1))
```

= WilhelmNaideGL(a, b, Cs)

```
WilhelmNaideGL = a * b * ((b - 1) / b * Cs) ^ (b - 1)
```

Mathematical Functions

= Asyg(a, b, c, d, m, x)

Equation for the asygmoidal function (entrainment, classification, etc)

Asyg = d + (a - d) / (1 + (x / c) ^ b) ^ m

= cubic_spline(input_column As Range, output_column As Range, x)

Given a data set consisting of a list of x values and y values, this function will smoothly interpolate a resulting output (y) value from a given input (x) value

Utilities Module

Sub FixAminproAddinRoot

Corrects for a known bug in Excel when Excel can't find the installation directory for the Aminpro Add-in after opening a file that was created by another user.

Sub StyleKill

Corrects a known bug in Excel that creates duplicate redundant style definitions. Will delete all redundant user-defined styles.

Sub ClearExcessRowsAndColumns

Corrects a known bug in Excel that causes the Excel file size to increase due to invisible formats in some of the worksheets.